**Nombres:**

**Daniel Rodríguez Manzanedo**

**Yeray Granada Layos**

**Miguel Ángel López Roche**

**María Gallego Martín**

**Carlos Gómez Robles**

**Ejercicio 1. Dado el siguiente Código de ordenación por el método de la burbuja:**

**public class BubbleSort {**

**/\*\***

**\* This method the BubbleSort method. Returns**

**\* the array in increasing order.**

**\***

**\* @param array The array to be sorted**

**\* @param size The count of total number of elements in array**

**\*\*/**

**public static <T extends Comparable<T>> void BS(T array[], int size) {**

**boolean swap;**

**int last = size - 1;**

**do**

**{**

**swap = false;**

**for (int count = 0; count < last-1; count++)**

**{**

**int comp = array[count].compareTo(array[count + 1]);**

**if (comp > 0)**

**{**

**T temp = array[count];**

**array[count] = array[count + 1];**

**array[count + 1] = temp;**

**swap = true;**

**}**

**}**

**} while (swap);**

**}**

**}**

**Al ejecutar el código con los valores de entrada obtenemos estos resultados:**

**[1 0 0 0 2 0 ] --> [0 0 0 1 2 0 ]**

**[8 7 6 5 4 9 ] --> [4 5 6 7 8 9 ]**

**Rellena la siguiente plantilla aplicando el método científico para depuración:**

|  |  |
| --- | --- |
| **Hipótesis** | **El código se ejecuta correctamente** |
| **Predicción** | **Al ejecutar el código recibiremos una salida** |
| **Experimento** | **Ejecutar el código** |
| **Observación** | **No ejecuta** |
| **Conclusión** | **Hipótesis rechazada** |

|  |  |
| --- | --- |
| **Hipótesis** | **Es necesario el método main para que funcione el código** |
| **Predicción** | **Al ejecutar el código recibiremos una salida** |
| **Experimento** | **Creamos el método main y ejecutamos el código** |
| **Observación** | **Se obtiene una salida** |
| **Conclusión** | **Hipótesis confirmada** |

|  |  |
| --- | --- |
| **Hipótesis** | **Al introducir un array desordenado, el programa nos devuelve el mismo array ordenado de forma ascendente** |
| **Predicción** | **Obtenemos el array ordenado** |
| **Experimento** | **Ejecutamos el código pasando el array [1 0 0 0 2 0 ]** |
| **Observación** | **El array no se encuentra ordenado de forma ascendente** |
| **Conclusión** | **Hipótesis rechazada** |

|  |  |
| --- | --- |
| **Hipótesis** | **El do while no funciona correctamente** |
| **Predicción** | **Al ejecutar el código obtenemos un array no ordenado de forma ascendente** |
| **Experimento** | **Ejecutamos el código utilizando el Debbuging para comprobar si el bucle so while recorre el array hasta que todos los elementos están ordenados de forma ascendente** |
| **Observación** | **El array no se encuentra ordenado pero no por fallo del do while** |
| **Conclusión** | **Hipótesis rechazada** |

|  |  |
| --- | --- |
| **Hipótesis** | **El for no recorre todos los elementos del array** |
| **Predicción** | **Si pasamos un array, el elemento que ocupa la última posición no es comparado** |
| **Experimento** | **Observar hasta que elemento del array recorre el for** |
| **Observación** | **No alcanza el último elemento del array** |
| **Conclusión** | **Hipótesis confirmada** |

|  |  |
| --- | --- |
| **Hipótesis** | **La variable last es la que produce el error** |
| **Predicción** | **Al ajustar last = size recorrerá todos los elementos del array** |
| **Experimento** | **Cambiamos last = size -1 por last = size** |
| **Observación** | **El for recorre todos los elementos del array** |
| **Conclusión** | **Hipótesis confirmada** |

**Ejercicio 2. Dado el siguiente Código fuente:**

**public class Goldbach {**

**/\*\***

**\* Checks if the number n is prime. Negative numbers and zero are declared**

**\* to be non-prime.**

**\***

**\* @param n**

**\* Number to be tested.**

**\* @return True if n is prime, false if not.**

**\*/**

**public static boolean isPrime(Integer n) {**

**for (int p = 2; p < Math.sqrt(n)-1; p++) {**

**if (n % p == 0) {**

**return false;**

**}**

**}**

**return true;**

**}**

**/\*\***

**\* Checks Goldbach conjecture for n, that is, checks if the even number n**

**\* can be written as the sum of two prime numbers.**

**\***

**\* If n is odd, it returs automatically true.**

**\***

**\* @param n**

**\* Number to be checked.**

**\* @return True if n can be decomposed as sum of prime numbers, false if**

**\* not. Also prints the decomposition if true.**

**\*/**

**public static boolean checkGoldbach(Integer n) {**

**for (int p = 1; p < n / 2; p++) {**

**if (isPrime(p) && isPrime(n - p)) {**

**System.out.println(n + "=" + p + " & " + (n - p));**

**return true;**

**}**

**}**

**return false;**

**}**

**}**

**Al ejecutar el código con los valores de entrada obtenemos estos resultados:**

**13=2 & 11  CORRECTO!**

**23=4 & 19  INCORRECTO!**

**Rellena la siguiente plantilla aplicando el método científico para depuración:**

|  |  |
| --- | --- |
| **Hipótesis** | **El código se ejecuta correctamente** |
| **Predicción** | **Al ejecutar, obtendremos una salida** |
| **Experimento** | **Ejecutamos el código** |
| **Observación** | **No se obtiene ninguna salida** |
| **Conclusión** | **Hipótesis rechazada** |

|  |  |
| --- | --- |
| **Hipótesis** | **Se necesita un método main para su correcto funcionamiento** |
| **Predicción** | **Obtendremos una salida** |
| **Experimento** | **Creamos un método main** |
| **Observación** | **Obtenemos una salida** |
| **Conclusión** | **Hipótesis aceptada** |

|  |  |
| --- | --- |
| **Hipótesis** | **El for del método checkGoldBach recorre todas las posibilidades** |
| **Predicción** | **For recorre todos los valores entre 1 y n/2 inclusive** |
| **Experimento** | **Ejecutar el método con n=2** |
| **Observación** | **El for no se ejecuta porque p=1 y y n/2=1 por lo que ya se cumpliría la condición de salida del for y, por lo tanto, nunca comprueba n/2** |
| **Conclusión** | **Hipótesis rechazada** |

|  |  |
| --- | --- |
| **Hipótesis** | **El for del método checkGoldBach recorre todas las posibilidades** |
| **Predicción** | **For recorre todos los valores entre 1 y n/2 inclusive** |
| **Experimento** | **En el for, cambiamos la condición de salida por p < N/2 +1 y ejecutar el método con n=2** |
| **Observación** | **Se entra al for y comprueba p=1 y p=N/2 (en este caso ambos p=1) que es el único valor posible para n=2** |
| **Conclusión** | **Hipótesis aceptada** |

|  |  |
| --- | --- |
| **Hipótesis** | **El for del método is prime recorre todas las posibilidades** |
| **Predicción** | **El For recorre los valores entre 2 y sqtr(n) inclusive** |
| **Experimento** | **Ejecutar el método con n=4** |
| **Observación** | **El for no se ejecuta porque p=2 y y sqtr(n)=2 por lo que ya se cumpliría la condición de salida del for sqtr(n) – 1 y, por lo tanto, nunca comprueban todos los valores** |
| **Conclusión** | **Hipótesis rechazada** |

|  |  |
| --- | --- |
| **Hipótesis** | **El for del método is prime recorre todas las posibilidades** |
| **Predicción** | **El For recorre los valores entre 2 y sqtr(n) inclusive** |
| **Experimento** | **Cambiar la condición de salida por p < Math.sqtr(n) + 1 y añadir en el if la condición de que n!=p. Ejecutamos el método con n=5** |
| **Observación** | **El for comprueba p=2 y sqtr(4), en este caso ambos p=2, el único divisor posible para este caso y realiza el método correctamente** |
| **Conclusión** | **Hipótesis aceptada** |

**Ejercicio 3.**

**Dada la siguiente implementación de una pila.**

**public class Stack {**

**private Integer[] arrayStack;**

**private Integer MAX\_SIZE = 5;**

**private Integer top;**

**/\*\***

**\* Create a stack LIFO with finite deep.**

**\*/**

**public Stack() {**

**this.arrayStack = new Integer[MAX\_SIZE];**

**this.top = 1;**

**}**

**/\*\***

**\* Inserts an element in the stack. If the stack is full, it shows a message**

**\* and stops**

**\***

**\* @param element**

**\* Element to be inserted.**

**\*/**

**public void push(Integer element) {**

**if (top >= MAX\_SIZE) {**

**System.out.println("The stack is full");**

**}**

**arrayStack[this.top] = element;**

**top++;**

**}**

**/\*\***

**\* Extracts an element from the stack. If the stack is empty, it shows a**

**\* message and returns -1.**

**\***

**\* @return Extracted element.**

**\*/**

**public Integer pull() {**

**if (top <= 0) {**

**System.out.println("The stack is empty");**

**return 0;**

**}**

**top--;**

**return arrayStack[top - 1];**

**}**

**/\*\***

**\* Returns the number of elements in the stack.**

**\***

**\* @return Number of elements in the stack.**

**\*/**

**public Integer getSize() {**

**return top;**

**}**

**/\*\***

**\* Extract nElements objects of the stack.**

**\***

**\* @param nElements**

**\* Number of elements to be extracted.**

**\* @return Array with the extracted elements.**

**\*/**

**public Integer[] pull(Integer nElements) {**

**Integer[] res = new Integer[nElements];**

**for (int i = 0; i < nElements; i++) {**

**res[i] = pull();**

**}**

**return res;**

**}**

**/\*\***

**\* Copy an stack into a new stack. The new stack has the same elements in**

**\* the same order as the old one.**

**\***

**\* @return Duplicated stack.**

**\*/**

**public Stack copyStack() {**

**Stack newStack = new Stack();**

**for (int i = 0; i < this.getSize(); i++) {**

**newStack.push(arrayStack[i]);**

**}**

**return newStack;**

**}**

**}**

**Al ejecutar el código obtenemos los siguientes resultados.**

**1.**

**Stack stack = new Stack();**

**stack.push(1);**

**stack.push(2);**

**stack.push(3);**

**stack.push(4); Correcto!!**

**2.**

**Stack stack = new Stack();**

**stack.push(12);**

**stack.push(2);**

**stack.push(43);**

**stack.push(95);**

**Integer v[] = stack.pull(stack.getSize());**

**Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Array index out of range: -1**

**at stack.Stack.pull(Stack.java:44)**

**at stack.Stack.pull(Stack.java:67)**

**at stack.Stack.main(Stack.java:97)**

**Siendo esta ejecución INCORRECTA**

**3.**

**Stack stack = new Stack();**

**stack.push(1);**

**stack.push(2);**

**stack.push(3);**

**stack.push(4);**

**Stack nueva = stack.copyStack();**

**The stack is full**

**Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Array index out of range: 5**

**at stack.Stack.push(Stack.java:30)**

**at stack.Stack.copyStack(Stack.java:86)**

**at stack.Stack.main(Stack.java:102)**

**Siendo esta ejecución INCORRECTA**

**4.**

**stack.push(1);**

**stack.push(2);**

**stack.push(3);**

**stack.push(4);**

**System.out.println("Elemento = " + stack.pull());**

**Elemento = 3**

**Siendo esta ejecución INCORRECTA**

**Aplique el método científico para su depuración rellenando la plantilla correspondiente.**

|  |  |
| --- | --- |
| **Hipótesis** | **El código se ejecuta correctamente** |
| **Predicción** | **Recibiremos una salida** |
| **Experimento** | **Runeamos el código** |
| **Observación** | **Produce error, falta un main** |
| **Conclusión** | **Hipótesis rechazada** |

|  |  |
| --- | --- |
| **Hipótesis** | **El código permite apilar valores en la pila** |
| **Predicción** | **Lo hará sin errores** |
| **Experimento** | **Introducimos los valores 1,2,3 y 4** |
| **Observación** | **El primer valor de la pila es null** |
| **Conclusión** | **Hipótesis rechazada** |

|  |  |
| --- | --- |
| **Hipótesis** | **El código permite apilar valores en la pila** |
| **Predicción** | **Lo hará sin errores** |
| **Experimento** | **Introducimos los valores 1,2,3 y 4 y cambiamos en el constructor top=1 por top= 0** |
| **Observación** | **Se introduce correctamente** |
| **Conclusión** | **Hipótesis aceptada** |

|  |  |
| --- | --- |
| **Hipótesis** | **No se pueden introducir más valores que el máximo permitido** |
| **Predicción** | **Saldrá un mensaje de error** |
| **Experimento** | **Introducimos 6 valores** |
| **Observación** | **Produce error ArrayIndexOutOfBounds** |
| **Conclusión** | **Hipótesis rechazada** |

|  |  |  |
| --- | --- | --- |
| **Hipótesis** | **No se pueden introducir más valores que el máximo permitido** |  |
| **Predicción** | **Saldrá un mensaje de error** |  |
| **Experimento** | **Introducimos 6 valores y añadimos un else a las sentecias posteriores al if** |  |
| **Observación** | **Se ejecuta correctamente** |  |
| **Conclusión** | **Hipótesis aceptada** |  |

|  |  |
| --- | --- |
| **Hipótesis** | **Es posible sacar un elemento de la pila** |
| **Predicción** | **Saldrá el último elemento** |
| **Experimento** | **A una pila con valores 1,2,3,4 le sacamos el valor 4** |
| **Observación** | **Devuelve el valor 3** |
| **Conclusión** | **Hipótesis rechazada** |

|  |  |  |
| --- | --- | --- |
| **Hipótesis** | **Es posible sacar un elemento de la pila** |  |
| **Predicción** | **Saldrá el último elemento** |  |
| **Experimento** | **A una pila con valores 1,2,3,4 le sacamos el valor 4 y cambiamos de orden las sentencias top—- y el número a devolver** |  |
| **Observación** | **Devuelve el valor 4** |  |
| **Conclusión** | **Hipótesis aceptada** |  |

|  |  |
| --- | --- |
| **Hipótesis** | **No es posible sacar un valor de una pila vacía** |
| **Predicción** | **Saldrá un mensaje explicativo y devolverá el número 0** |
| **Experimento** | **Intentamos sacar un valor de una pila vacía** |
| **Observación** | **Lo ejecuta correctamente** |
| **Conclusión** | **Hipótesis aceptada** |

|  |  |
| --- | --- |
| **Hipótesis** | **Se puede sacar elementos de uno en uno de un array y meterlo en otra** |
| **Predicción** | **Saldrán en el orden contrario en la segunda array** |
| **Experimento** | **Ejecutamos** stack.pull(stack.getSize()); |
| **Observación** | **Se ejecuta tal cual se predice** |
| **Conclusión** | **Hipótesis aceptada** |

|  |  |
| --- | --- |
| **Hipótesis** | **Es posible copiar una pila** |
| **Predicción** | **Al copiar una pila tendrá los mismos valores que la otra** |
| **Experimento** | **Copiamos la pila 1,2,3,4 con el método** stack.copyStack(); |
| **Observación** | **Lo ejecuta correctamente** |
| **Conclusión** | **Hipótesis aceptada** |

**Viendo la cantidad de errores que hay, prepare una batería de test lo suficientemente completa para validar el correcto funcionamiento de la pila.**