Preguntas teóricas Listado 1 Estructuras de datos.

1. **Entrega el siguiente código comentado con los resultados a la derecha de cada System.out.println() y contesta:**
    a. **¿Cómo se comparan dos cadenas?:** Se comparan con el método equals();
    b. **¿Para qué se utiliza el operador de comparación == entre objetos?:** Para comparar si son el mismo objeto.
    c. **Una cadena puede crearse mediante new o directamente con su literal. ¿Se crean de igual manera? ¿Se aprovecha la memoria de alguna manera?:** No, cuando creamos una cadena mediante new se crea una nueva cadena diferente a las demás mientras que cuando la creamos directamente con su literal aprovechamos la memoria ya que no se crea una nueva cadena sino que se reutiliza.
2. **Crea el siguiente código. Contesta a las siguientes preguntas:**
    a. **¿Cuándo la comparación de dos referencias es true?:** Cuando ambas apuntan al mismo trozo de memoria.
    b. **¿Siempre se inicializa una referencia a null?:** Siempre y cuando sea static y se inicialice por defecto.
    c. **¿Por qué vector1 y vector2 han de ser static? ¿Cuál sería otra solución?:** Porque se invocan en un método static, otra solución sería inicializarlos a null o crearlos dentro del main directamente.
3. **Indica las clases Wrappers o envoltorios correspondientes a los siguientes datos primitivos:**

Tipo primitivo	Clase envoltorio (Wrapper)
boolean	Boolean
byte	Byte
char	Character
float	Float
int	Integer
long	Long
short	Short

5. **Un iterador es un objeto que se implementa en la interfaz Iterator o ListIterator. Se utiliza para el recorrido, obtención o modificación de los elementos de una colección. Basándote en el siguiente código y analizando el interfaz Iterator indica la descripción. y signatura de los siguientes métodos: hasNext() y next()**
    - **hasNext():** Signatura: boolean hasNext(). Devuelve true si el iterador encuentra un elemento delante de su posición, de no ser así lanzará una excepción.
    - **next():**Signatura: E next(). Obtiene el elemento siguiente en la iteración.

6. **Continuando con la clase anterior "TestIterator", analiza el siguiente código y contesta:**
    a. **Analiza el interfaz ListIterator e indica la descripción y signatura de los siguientes métodos: hasNext(), hasPrevious(), next() y previous()**
        i. **hasNext():** Signatura: boolean hasNext(). Devuelve true si encuentra un elemento siguiente.
        ii. **hasPrevious():** Signatura: boolean hasPrevious(). Devuelve true si encuentra un elemento anterior, iterando hacia atrás.
        iii. **next():** Signatura: E next(). Devuelve el elemento en la lista y avanza la posición del cursor.
        iv. **previous():** Signatura: E previous(). Devuelve el elemento previo en la lista y mueve la posición del cursor.
    b. **Analiza el interfaz ListIterator e indica la descripción y signatura del método set(). Identifica los requisitos para su uso.**
        i. **set():** Signatura: void set(E e). Reemplaza el último elemento recorrido por next() o previous() con el elemento especificado entre paréntesis.
7. **Continuando con la clase anterior "TestIterator", analiza el siguiente código y contesta:**
    a. **Indica los métodos implicados en el recorrido hacia delante:** hasNext() y next().
    b. **Indica los métodos implicados en el recorrido hacia atrás:** hasPrevious() y previous().
    c. **¿Qué sucedería en el siguiente código si el listIterator no partiera del final de la colección en el recorrido hacia atrás?:** Que recorrería desde donde este el puntero.
    d. **Averigua la forma de posicionar el iterador al final de la colección:** Reiniciando el iterador volviendo a llamar al método listIterator(arrayList.size()) y pasándole por parámetro la última posición del cursor en el arrayList creado anteriormente.

```
System.out.println("Reiniciado y recorriendo hacia atrás..");
listIterator = arrayList.listIterator(arrayList.size());// Reiniciando
                                                        // llamando al
                                                        // metodo
                                                        // listIterator
                                                        // y pasandole
                                                        // por parametro
                                                        // la ultima
                                                        // posicion del
                                                        // cursor en el
                                                        // arrayList.
while (listIterator.hasPrevious()) {// 10. Bucle con hasPrevious();
    Object element = listIterator.previous();// 11. Dentro del bucle,
                                             // obtención del
                                             // elemento mediante
                                             // previous();

    System.out.print(element + "\t");
}
```

8. **Implementa el siguiente trozo de código y contesta:**
    a. **¿Qué estructura de datos se instancia en este código?:** Se instancia una lista de tipo ArrayList.
    b. **¿En qué paquete se encuentra?:** Se encuentra en el paquete java.util.
    c. **Localiza el interfaz java.util.Collection que implementa e indica al menos tres métodos implementados por la clase (signatura y descripción):**
        i. **isEmpty():**Signatura: boolean isEmpty(). Devuelve true si la Collection está vacía.
        ii. **add(E e):** Signatura: boolean add(E e). Añade un objeto a la Collection, devolviendo true si ha conseguido añadirlo.
        iii. **remove(Object o):** Signatura: boolean remove(Object o). Elimina un objeto de la Collection, devolviendo true si ha conseguido eliminarlo.
    d. **¿Se permiten elementos duplicados?:** Dependiendo de si la collection lo permite o no.
    e. **¿Hay que saber su tamaño en su creación?:** No se necesita saber el tamaño de la Collection en el momento de su creación, simplemente se va ampliando conforme se le vaya añadiendo elementos a la misma.
    f. **¿Es una estructura dinámica o estática?:** Es una estructura dinámica ya que aumentara su tamaño conforme le añadamos elementos.
    g. **¿Es una estructura homogénea o heterogénea?:** Es una estructura homogénea cuando se le indica el tipo que contendrá, de lo contrario será heterogénea.
    h. **¿Se utiliza el autoboxing?:** Si, cuando se añade add(1) se convierte en Integer ya que es un objeto.
    i. **¿Existe algún Wrapper?:** con el new Double() y el new Double().
9. **Continuando con la clase anterior "TestArrayList", analiza el siguiente código y contesta:**
    a. **La palabra reservada "instanceof" es un operador especial que compara un objeto con un tipo. Se puede utilizar para comprobar si un objeto es una instancia de una clase, una instancia de una subclase, o una instancia de una clase que implementa una interfaz concreta. En el código, ¿dónde se utiliza?¿de qué tipo es el objeto? ¿Cómo se introdujo el elemento en la estructura?:** Se utiliza cuando: arrayList.get(0) instanceof Integer, el objeto es de tipo entero, se introdujo mediante add y se le pasaba por parámetro el número de tipo int.
    b. **El método iterator(), ¿para qué se utiliza?¿Existe otra manera de hacerlo?:** Se utiliza para crear un iterador sobre ese ArrayList, existe otra manera que sería convirtiendo el ArrayList en un Array y recorrerlo con un bucle for mejorado.
    c. **Indica la signatura de los métodos utilizados para:**
        i. **Añadir:** add(E e).
        ii. **Eliminar:** remove(Object o) ó remove(int index).
        iii. **Obtener:** get(int index, E element).
        iv. **Convertir a estructura estática:** toArray().
        v. **Averiguar:**
            1. **Tamaño:** size().
            2. **Si la estructura está vacía:** isEmpty().

3. **Si un elemento está en la estructura:** contains().
4. **Posición de un elemento:** indexOf().