Examen 111Mil – Control de Stock - Resolución

En el comando central de 111Mil se concentran todos los elementos necesarios para el dictado y promoción de los cursos, incluyendo las computadoras con las que se realizan las actividades prácticas. Debido a la cantidad de computadoras, el encargado del almacenamiento solicita el desarrollo de un sistema de control de stock.

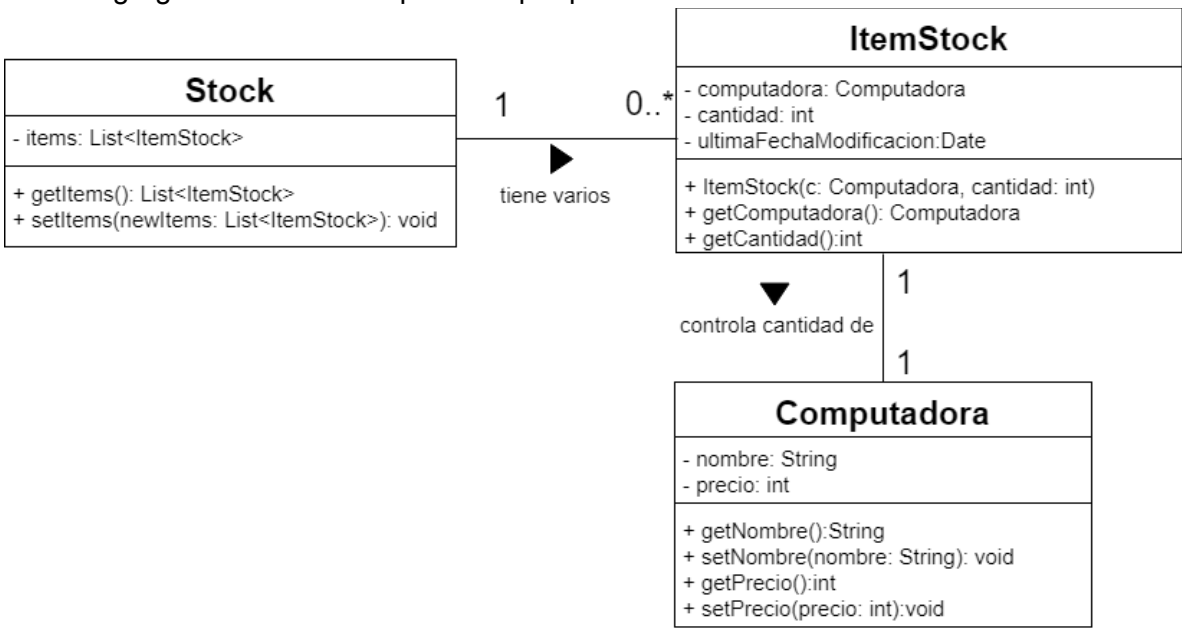
De acuerdo a lo comentado por el encargado, las computadoras almacenadas tienen un nombre, por ejemplo: laptop, all-in-one, pc en miniatura, etc; y un precio asociado. Para cada Computadora, se debe registrar la cantidad existente en el almacén.

Al encargado le interesa, por sobre todas las cosas, conocer aquellos ítems que se encuentran por debajo de cierto límite (para realizar reposiciones), y conocer qué computadora es la más cara (para tomar medidas de seguridad). El equipo de desarrollo, del cual Ud. es parte, acepta el desafío y se embarca en la creación del sistema de control de stock, comenzando por el diagrama de clases.

Ejercicio 1. Implementar desde el diagrama de clases

Parte del equipo de desarrollo ya comenzó con la implementación del sistema de control de stock y se nos pide completar el código creado hasta el momento. **A partir del diagrama de clases y el código dado, complete:**

A. La declaración de las variables: computadora y cantidad en la clase ItemStock, nombre en la clase Computadora.
B. El método getItems de la clase Stock, el cual retorna una copia de la lista interna de ítems.
C. El método setItems de la clase Stock, el cual limpia la lista interna de ítems y agrega todos los items pasados por parámetro.



```java
public class ItemStock {
    // A: Agregar variables computadora y cantidad
    private Computadora computadora;
    private int cantidad;

    private Date ultimaFechaModificacion;
    public ItemStock(Computadora c, int cantidad) {
        this.computadora = c;
        this.cantidad = cantidad;
    }
    public Computadora getComputadora() {
        return computadora;
    }
    public int getCantidad() {
        return cantidad;
    }
}

public class Computadora {
    // A: Agregar la variable nombre
    private String nombre;

    private int precio;
    public int getPrecio() {
        return precio;
    }
    public void setPrecio(int precio) {
        this.precio = precio;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}

public class Stock {
    private List<ItemStock> items = new ArrayList<>();

    // B: Agregar método getItems, retornando una copia de items.

    // C: Agregar método setItems, copiando los elementos en newItems

    public List<ItemStock> getItems(){
      return new ArrayList<ItemStock>(this.items);
```

```
    }
    public void setItems(List<ItemStock> newItems){
      this.items = new ArrayList<ItemStock>(newItems);
    }
}
```

Ejercicio 2. Implementar un método a partir de un enunciado

Para satisfacer uno de los requerimientos del encargado, se pide crear en la clase **Stock** el método **consultarItemsFaltantes**, con la siguiente signatura:

```
public List<ItemStock> consultarItemsFaltantes(int cantidadMáxima) {
}
```

El mismo retorna una **nueva lista de ítems**, pero sólo incluyendo aquellos **ItemStock** con **cantidad menor al valor indicado por el usuario**.

Afortunadamente, el equipo nos ha dejado la siguiente lista de tips para implementar el método:

- Puede crear una lista auxiliar utilizando el siguiente código:

  List<ItemStock> resultado = new ArrayList<ItemStock>()
- Debe agregar cada ItemStock (de la lista "items") al resultado solo si cumplen la condición dada.
- Recuerde, ¡no retorne del método sin antes haber recorrido la totalidad de la lista!

```
public List<ItemStock> consultarItemsFaltantes(int cantidadMáxima) {
      List<ItemStock> resultado = new ArrayList<ItemStock>();
      for (ItemStock item: this.items)
            if (item.getCantidad() <= cantidadMáxima)
                  resultado.add(item);
      return resultado;
}
```

Ejercicio 3. Identificar error en código.

Un colega nos acerca una porción de código que no hace lo que debería hacer y nos solicita ayuda. El código en cuestión es un pequeño algoritmo para buscar la Computadora más costosa, de manera que el encargado del almacén pueda guardarla en un lugar seguro.

```
public Computadora obtenerComputadoraMasCostosa() {
  Computadora masCostosa = null;
  for (ItemStock item : this.items) {
    if (masCostosa == null
      || item.getComputadora().getPrecio() <= masCostosa.getPrecio())
      masCostosa = item.getComputadora();
```

```
  return masCostosa;
 }
```

Por suerte, nuestro colega ya identificó posibles errores y solo debemos indicar cual de ellos **(solo uno)** es el que genera el problema.

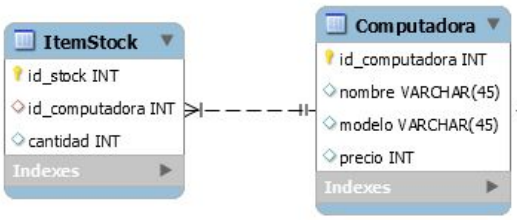
1. La variable “masCostosa” debería estar inicializada con una Computadora con precio 0.
2. Debería utilizar un while en lugar de un for.
3. **La comparación entre el precio de cada item y el de la variable “masCostosa” debería ser mayor estricto (>) o mayor o igual (>=).**
4. El retorno de la función debería ser “this.items”.

Indique la opción correcta: _

Indique la opción correcta: 3

Ejercicio 4. Interpretación de DER.

Luego de un tiempo, el sistema evolucionó de acuerdo a los pedidos del encargado. Además de registrar las cantidades de cada Computadora, se registra cada acción de agregar o quitar del Stock utilizando un historial de acciones. En el caso de acciones de tipo “quitar”, se registra a qué curso fueron destinadas las Computadoras. En el caso de “agregar”, se registra el Proveedor que agregó Computadoras. El diagrama de entidades y relaciones (DER) del sistema actualizado es el siguiente:



A partir del DER, **responda Verdadero (V) o Falso (F)** a las siguientes afirmaciones:

En ItemStock, la tabla tiene una clave compuesta	
Cada ItemStock tiene, como máximo, una sola computadora asociada	
El valor 200 es un valor válido para la columna “cantidad” de ItemStock	
La relación entre ItemStock y Computadora es N a 1	
La clave principal de Computadora es el id_computadora	

id_computadora es una clave foránea de ItemStock	
La relación entre ItemStock y Computadora es 1 a N	
“159 pesos” es un valor válido para el precio de Computadora	

En ItemStock, la tabla tiene una clave compuesta	**F**
Cada ItemStock tiene, como máximo, una sola computadora asociada	**V**
El valor 200 es un valor válido para la columna “cantidad” de ItemStock	**V**
La relación entre ItemStock y Computadora es N a 1	**V**
La clave principal de Computadora es el id_computadora	**F**
id_computadora es una clave foránea de ItemStock	**V**
La relación entre ItemStock y Computadora es 1 a N	**F**
“159 pesos” es un valor válido para el precio de Computadora	**F**

Ejercicio 5. Completando consultas

El encargado del depósito desea averiguar la cantidad de unidades disponibles de aquellas computadoras cuyo precio que supera los 50000 pesos. Los datos que necesita el encargado son:

- **Nombre de la computadora**
- **Cantidad en el stock**
- **Precio de la computadora**

Basándose en el DER del ejercicio 4, realice una consulta SQL de acuerdo a lo solicitado por el encargado.

```
SELECT      computadora.nombre,
            computadora.precio,
            itemstock.cantidad
FROM Computadora computadora
JOIN ItemStock itemstock
ON itemstock.id_computadora = computadora.id_computadora
WHERE computadora.precio > 50000
```