Nombre:..... ***Legajo:***.....

Segundo Parcial de Programación Orientada a Objetos

14 de Junio de 2010

Ej. 1	Ej. 2	Ej. 3	Nota

- **Condición de aprobación: Tener dos ejercicios calificados como B o B-.**
- **Los ejercicios que no se ajusten estrictamente al enunciado, no serán aceptados.**
- **No es necesario agregar las sentencias import**
- **Además de las clases pedidas se pueden agregar las que se consideren necesarias.**

Ejercicio 1

Desarrollar la clase **IgnoreNullsList** que implementa la interface **List** (la ofrecida por el API de Java) y agrega un método **ignoreNullsIterator** que devuelve un iterador que ignora los elementos nulos de la lista.

Lo que sigue es un ejemplo de uso de la nueva lista:

IgnoreNullsListTest.java

```
public class IgnoreNullsListTest {

	public static void main(String[] args) {
		IgnoreNullsList<Integer> list = new IgnoreNullsList<Integer>();

		list.add(1);
		list.add(null);
		list.add(2);
		list.add(5);
		list.add(null);
		list.add(3);
		list.add(null);

		Iterator<Integer> it = list.ignoreNullsIterator();
		while(it.hasNext()) {
			Integer num = it.next();
			System.out.println(num);
			if (num == 2)
				it.remove();
		}
		System.out.println("--------------------------");

		for (Integer i: list) {
			System.out.println(i);
		}
	}
}
```

Salida de la ejecución del programa anterior:

```
1
2
5
3
--------------------------
1
null
5
null
3
null
```

Consideraciones:

- El método **remove** elimina el elemento que la última invocación a **next** haya retornado.
- En caso de invocar a **remove** sin haber invocado a **next** previamente se lanza una **IllegalStateException**.
- No es posible invocar dos veces consecutivas a **remove**.
- El método **next** lanza una **NoSuchElementException** cuando se ya se ha iterado por todos los elementos (y por lo tanto una invocación a **hasNext** hubiera retornado false).

Ejercicio 2

Se cuenta con la clase **CoffeeMachine** que modela una expendedora de café. Dicha clase considera que una máquina tiene parametrizado (en el constructor) el precio de un café. Además ofrece los métodos **loadCoffee** y **loadCoins** que permiten recargar unidades de café y de monedas respectivamente.

Cuando un usuario quiere comprar un café, debe agregar monedas a través una o más invocaciones al método **insertCoin**. En caso que la máquina no disponga de unidades de café, esta operación lanza la excepción **NotEnoughCoffeeException.** De lo contrario, acumula las monedas.

Cuando se ha acumulado el importe correcto, el usuario debe retirar el café invocando al método **takeCoffee**. En caso que el importe no fuera suficiente, el método lanza la excepción **NotEnoughMoneyException.** Si no tuviera las monedas necesarias para dar el vuelto, lanza la excepción **NoChangeException** y anula el pedido, permitiendo que el usuario cancele la transacción a través del método **cancelOperation**.

Esta clase utiliza la interface **CoinManager** y una implementación de dicha interface para la administración de las monedas de la máquina. Las monedas están modeladas mediante el enumerativo **Coin**.

CoffeeMachine.java

```
public class CoffeeMachine {

	private int price;
	private int coffeeAmount;
	private int currentMoney;
	private CoinManager coinManager = new CoinManagerImpl();

	public CoffeeMachine(int price) {
		this.price = price;
	}

	public void loadCoffee(int amount) {
		this.coffeeAmount += amount;
	}

	public void loadCoins (List<Coin> coins) {
		for (Coin c: coins) {
			this.coinManager.add(c);
		}
	}

	public void insertCoin(Coin coin) throws NotEnoughCoffeeException{
		if (this.coffeeAmount == 0) {
			throw new NotEnoughCoffeeException();
		} else {
			coinManager.add(coin);
			currentMoney += coin.getValue();
		}
	}

	public List<Coin> takeCoffe() throws NoChangeException, NotEnoughMoneyException {
		if (currentMoney >= price) {
			int change = currentMoney - price;
			List<Coin> list = coinManager.releaseCoins(change);
			this.coffeeAmount--;
			currentMoney = 0;
			return list;
		}
		throw new NotEnoughMoneyException();
	}

	public List<Coin> cancelOperation() {
		try {
			int change = currentMoney;
			currentMoney = 0;
			return coinManager.releaseCoins(change);
		} catch (NoChangeException e) {
			return null;
		}
	}

	public int getCoffeeAmount() {
		return coffeeAmount;
	}

	public int getCurrentMoney() {
		return currentMoney;
	}

	public int getPrice() {
		return price;
	}
}
```

Coin.java

```
public enum Coin {

	ONE_DOLLAR(100), FIFTY_CENTS(50), TWENTYFIVE_CENTS(25), TEN_CENTS(10), FIVE_CENTS(5);

	private int value;

	private Coin(int value) {
		this.value = value;
	}

	public int getValue() {
		return value;
	}
}
```

CoinManager.java

```
import java.util.List;

public interface CoinManager {

	/**
	 * Agrega una moneda al repositorio de monedas
	 * @param coin Moneda a agregar
	 */
	public void add(Coin coin);

	/**
	 * Devuelve una lista de monedas que suman la cantidad solicitada.
	 * Las mismas se sacan del repositorio
	 * @param value Importe solicitado
	 * @throws NoChangeException Cuando no tiene monedas suficientes para el importe solicitado
	 */
	public List<Coin> releaseCoins(int value) throws NoChangeException;
}
```

Lo que sigue es un ejemplo del **CoffeeMachine** simulando las acciones del usuario.

CoffeeMachineTest.java

```
public class CoffeMachineTest {

   public static void main(String[] args) {

	CoffeeMachine machine = new CoffeeMachine(75);
	List<Coin> coins = new ArrayList<Coin>();
	coins.add(Coin.TEN_CENTS);
	coins.add(Coin.TEN_CENTS);
	coins.add(Coin.FIVE_CENTS);
	coins.add(Coin.TWENTYFIVE_CENTS);

	machine.loadCoins(coins);

	List<Coin> change;

	System.out.println("----------------------------"); //Aún no se ha cargado café.
	try {
		machine.insertCoin(Coin.ONE_DOLLAR);           //Lanza excepción por no tener café.
	} catch (Exception e) {
		System.out.println(e.getClass().getSimpleName());
		machine.cancelOperation();
	}

	System.out.println("----------------------------");
	machine.loadCoffee(5);                              //Se cargan 5 unidades de café.

	System.out.println("----------------------------");
	try {
		machine.insertCoin(Coin.FIFTY_CENTS);         //Se ingresan monedas.
		machine.insertCoin(Coin.FIFTY_CENTS);
		change = machine.takeCoffe();                 //Se pide un café y se recibe
		System.out.println("Change:" + change);       // el vuelto.
	} catch (Exception e) {
		System.out.println(e.getClass().getSimpleName());
		machine.cancelOperation();
	}

	System.out.println("----------------------------");
	try {
		machine.insertCoin(Coin.FIFTY_CENTS);         //Se ingresan monedas.
		machine.insertCoin(Coin.FIFTY_CENTS);
		machine.insertCoin(Coin.FIFTY_CENTS);         //Esta moneda sobra, pero se acepta.
		change = machine.takeCoffe();                 //Se pide un café y se recibe el vuelto.
		System.out.println("Change:" + change);
	} catch (Exception e) {
```

```
            System.out.println(e.getClass().getSimpleName());
            machine.cancelOperation();
      }

      System.out.println("----------------------------");
      try {
            machine.insertCoin(Coin.ONE_DOLLAR);          //Se ingresa una moneda.
            change = machine.takeCoffe();                 //Se pide un café pero la operación
      } catch (Exception e) {                             //lanza excepción por falta de cambio.
            System.out.println(e.getClass().getSimpleName());
            machine.cancelOperation();                    // se cancela la operación.
      }

      System.out.println("----------------------------");
      try {
            machine.insertCoin(Coin.FIVE_CENTS);          //Se ingresa una moneda insuficiente
            change = machine.takeCoffe();                 // Se pide un café pero la operación
      } catch (Exception e) {                          // lanza excepción por dinero insuficiente
            System.out.println(e.getClass().getSimpleName());
            machine.cancelOperation();                    // se cancela la operación.
      }
   }
}
```

La siguiente es la salida de la ejecución del programa anterior:

```
----------------------------
NotEnoughCoffeeException
----------------------------
----------------------------
Change:[TWENTYFIVE_CENTS]
----------------------------
Change:[FIFTY_CENTS, TEN_CENTS, TEN_CENTS, FIVE_CENTS]
----------------------------
NoChangeException
----------------------------
NotEnoughMoneyException
```

Dado que la información que recibe un usuario de esta máquina es casi nula, se desea implementar un máquina con un visor que informe las diferentes situaciones. Para ello se pide implementar **MessageCoffeeMachine** que responde a las mismas operaciones que la otra máquina pero además brinda información al usuario (escribiendo mensajes en un **PrintStream**). Lo que sigue es un programa que muestra el uso de **MessageCoffeeMachine**, con las mismas situaciones que **CoffeeMachine,** donde los comentarios muestran en texto impreso en el **PrintStream**:

```
public class MessageCoffeMachineTest {

   public static void main(String[] args) {

        MessageCoffeeMachine machine = new MessageCoffeeMachine(75, System.out);
                                                                  // CoffeMachine Empty
        List<Coin> coins = new ArrayList<Coin>();
        coins.add(Coin.TEN_CENTS);
        coins.add(Coin.TEN_CENTS);
        coins.add(Coin.FIVE_CENTS);
        coins.add(Coin.TWENTYFIVE_CENTS);

        machine.loadCoins(coins);
        List<Coin> change;

        System.out.println("----------------------------");
        machine.insertCoin(Coin.ONE_DOLLAR);        // Sorry, but the CoffeeMachine is Empty.

        System.out.println("----------------------------");
        machine.loadCoffee(5);                      // Insert coins ...


        System.out.println("----------------------------");
        machine.insertCoin(Coin.FIFTY_CENTS);       // 25 cents left.
        machine.insertCoin(Coin.FIFTY_CENTS);       // Please, take your coffee.
        change = machine.takeCoffe();              // Insert coins ...

        System.out.println("Change:" + change);     // Change:[TWENTYFIVE_CENTS]

        System.out.println("----------------------------");
        machine.insertCoin(Coin.FIFTY_CENTS);       // 25 cents left
        machine.insertCoin(Coin.FIFTY_CENTS);       // Please, take your coffee
        machine.insertCoin(Coin.FIFTY_CENTS);       // Please, take your coffee
        change = machine.takeCoffe();               // Insert coins ...

        System.out.println("Change:" + change);     // Change:[FIFTY_CENTS, TEN_CENTS, TEN_CENTS,
                                                                                  FIVE_CENTS]
```

```
        System.out.println("----------------------------");
        machine.insertCoin(Coin.ONE_DOLLAR);        // Please, take your coffee
        change = machine.takeCoffe();               // I have no change. Please cancel operation.
        machine.cancelOperation();                  // Insert coins ...

        System.out.println("----------------------------");
        machine.insertCoin(Coin.FIVE_CENTS);        // 70 cents left.
        change = machine.takeCoffe();               // I said: 70 cents left.
    }
}
```

Ejercicio 3

Escribir una implementación de la interface **`Multimap<K,V>`** que modela un mapa donde la clave de tipo **K**, tiene uno o más valores asociados del tipo **V**.

```
public interface Multimap<K,V> {

        /**
         * Asocia la clave key con el objeto value
         * @param key Clave
         * @param value Valor asociado
         */
        public void put(K key, V value);

        /**
         * Obtiene todos los objetos asociados a la clave key
         * @param key Clave buscada
         * @return La colección de objetos o null si la clave no existe
         */
        public Collection<V> get(K key);

        /**
         * Devuelve el tamaño de la colección
         */
        public int size();

        /**
         * Elimina la asociación entre key y value. Si no existe la asociación no hace nada.
         * @param key Clave buscada
         * @param value Valor asociado
         */
        public void remove(K key, V value);

        /**
         * Elimina todas las asociaciones de la clave key.
         * @param key Clave buscada
         */
        public void remove(K key);
}
```