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

**Recuperatorio del Segundo Parcial de Programación Orientada a Objetos**

1 de Julio 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*

Se cuenta con la clase **MessageCoffeeMachine**, que permite mostrar mensajes que detallan el proceso de la compra de un café. Dicha clase recibe en el constructor un objeto que implementa la interface **Display**, que ofrece un método para registrar mensajes. Se tiene una implementación de **Display**, la clase **ConsoleDisplay**, que envía los mensajes a la salida estándar.

MessageCoffeeMachine.java

```
public class MessageCoffeeMachine extends CoffeeMachine{

	private Display display;

	public MessageCoffeeMachine(int price, Display display) {
		super(price);
		this.display = display;
		display.showMessage("CoffeMachine Empty");
	}

	@Override
	public void insertCoin(Coin coin) {
		try {
			super.insertCoin(coin);
			int left = getPrice() - getCurrentMoney();
			if (left > 0) {
				display.showMessage(left + " cents left.");
			} else {
				display.showMessage("Please, take your coffee");
			}
		} catch(NotEnoughCoffeeException e) {
			display.showMessage("Sorry, but the CoffeeMachine is Empty.");
		}
	}

	. . .
}
```

Display.java

```
public interface Display {

	public void showMessage(String message);

}
```

ConsoleDisplay.java

```
public class ConsoleDisplay implements Display {

	@Override
	public void showMessage(String message) {
		System.out.println(message);
	}
}
```

Se necesita enviar mensajes a más de un dispositivo en forma simultánea, es decir que se va a contar con más de una implementación de la interface **Display** y se pretende que el mensaje llegue a cada una de estas implementaciones.
**Se pide implementar lo necesario para que esto sea posible**.
Mostrar el uso de lo implementado en el siguiente ejemplo, que en este caso cuenta con tres implementaciones de Display (NO DEBEN ESCRIBIRLAS). Completar la zona punteada con lo que sea necesario:

```
public class MessageCoffeMachineTest {

	public static void main(String[] args) {

		Display console = new ConsoleDisplay();
		Display lcd = new LCDDisplay();
		Display file = new FileDisplay();

		.................................................
		.................................................

		MessageCoffeeMachine machine = new MessageCoffeeMachine(75, display);

		machine.insertCoin(Coin.ONE_DOLLAR);
		/* Esto imprime en console , en el display LCD y en un archivo
		 * el mensaje "CoffeMachine Empty" */
	}
}
```

**Ninguno de los tres archivos que se muestran arriba (MessageCoffeeMachine, Display, ConsoleDisplay) pueden ser modificados.**

### *Ejercicio 2*

Se quiere contar con un tipo de lista que permita aplicar una función **`injectInto`** similar a la que ofrece Smalltalk. Dicha función recibe un parámetro inicial de tipo S llamado *semilla* y una función (en Smalltalk sería un bloque) **f: S,T → S**. El tipo T corresponde al tipo de los elementos de la lista. La función se aplica la primera vez sobre la semilla y el primer elemento de la lista. Esto genera un resultado de tipo S que se convierte en la próxima semilla y así se opera sobre el resto de los elementos de la lista. El valor de retorno corresponde a la semilla generada al procesar el último elemento de la lista.

Se pide escribir la interface **`InjectableList<T>`** que extiende a **`List<T>`**, agregando el método **`injectInto`**. Escribir además una implementación de **`InjectableList<T>.`** Escriba las clases y/o interfaces extras que considere necesarias.

### *Ejercicio 3*

Se cuenta con un conjunto de clases que modelan el proceso de envío de mensajes de texto. La clase **PhoneCentral** es la central que permite la designación de números de teléfono. La clase **CellPhone** modela el teléfono y la clase **Message** modela el mensaje.

PhoneCentral.java

```
public class PhoneCentral {

	private Map<String, CellPhone> phones = new HashMap<String, CellPhone>();

	public PhoneCentral() {
	}

	public CellPhone newPhone(String number) {
		if (getPhone(number) != null) {
			return null;
		}
		CellPhone cellphone = new CellPhone(number, this);
		phones.put(number, cellphone);
		return cellphone;
	}

	private CellPhone getPhone(String number) {
		return phones.get(number);
	}

	public void routeMessage(Message message) {
		CellPhone phone = phones.get(message.getTo());
		if (phone != null) {
			phone.receiveMessage(message);
		} else {
			throw new UnknownRecipientException();
		}
	}

}
```

CellPhone.java

```
public class CellPhone {

	private String number;
	private PhoneCentral central;

	public CellPhone(String number, PhoneCentral central) {
		super();
		this.number = number;
		this.central = central;
	}
	public void receiveMessage(Message message) {
		System.out.println("FROM: " + message.getFrom() +
				" TO: " + message.getTo() + " MSG: " + message.getText());
	}
	public void sendMessage(String to, String text) {
		central.routeMessage(new Message(number, to, text));
	}
}
```

Message.java

```
public class Message {

	private String from;
	private String to;
	private String text;

	public Message(String from, String to, String text) {
		super();
		this.from = from;
		this.to = to;
		this.text = text;
	}

	public String getFrom() {
		return from;
	}

	public String getTo() {
		return to;
	}

	public String getText() {
		return text;
	}
}
```

**Se pide implementar una central** que sea capaz de registrar estadísticas de envío de mensajes. Las estadísticas corresponden a cantidad de caracteres enviados, cantidad de mensajes enviados y cantidad de mensajes fallidos (por error en el número destino). Lo que sigue es un programa de ejemplo de la nueva central:

```
public class TestPhones {

	public static void main(String[] args) {

		EnhancedPhoneCentral central = new EnhancedPhoneCentral();

		CellPhone[] phones = new CellPhone[4];
		for (int i=0; i< 4; i++) {
			phones[i] = central.newPhone("111" + i);
		}
		try {
			phones[0].sendMessage("1111", "Hola");
			phones[1].sendMessage("1110", "Que tal?");
			phones[2].sendMessage("1110", "Buen dia");
			phones[2].sendMessage("1117", "Buen dia");
		} catch(UnknownRecipientException e){
			System.out.println("Alguna llamada falló");
		}
		central.showStats();
	}
}
```

Salida del programa anterior:

```
FROM: 1110 TO: 1111 MSG: Hola
FROM: 1111 TO: 1110 MSG: Que tal?
FROM: 1112 TO: 1110 MSG: Buen dia
Alguna llamada falló
Teléfono: 1110 Caracteres enviados: 4 - Mensajes enviados: 1 -  Mensajes fallidos: 0
Teléfono: 1111 Caracteres enviados: 8 - Mensajes enviados: 1 -  Mensajes fallidos: 0
Teléfono: 1112 Caracteres enviados: 8 - Mensajes enviados: 1 -  Mensajes fallidos: 1
Teléfono: 1113 Caracteres enviados: 0 - Mensajes enviados: 0 -  Mensajes fallidos: 0
```