# Sesión 17: Simulacro Ejercicio Evaluable

## Hoja de problemas

Programación 2

Ángel Herranz
aherranz@fi.upm.es
Universidad Politécnica de Madrid

Abril 2019

Este ejercicio debe servir como preparación para la realización del Ejercicio Entregable en el Grupo 1M el día 9 de abril de 2019.

## Normas

- El ejercicio puede realizarse por parejas o individualmente.
- El ejercicio dura 1 hora y 30 minutos.
- **Lee todo el enunciado** antes de empezar.
- La entrega deberá realizarse a través de la tarea Moodle indicada por el profesor: "Simulacro ejercicio evaluable (4 de abril)".
- Junto a este PDF habrás encontrado un fichero *.zip* con esquemas de código que te ayudarán a realizar el ejercicio. Mi recomendación es que lo descomprimas en un directorio (por ejemplo `sesion17`) y trabajes en él.
- Al descomprimirlo verás dos directorios (`./src` y `./bin`), dentro de `./src` podrás encontrar código de apoyo que te servirá como punto de partida.
- Tendrás que entregar cuatro ficheros *.java*:
  - `src/TestsListaAcotada.java`
  - `src/acotados/ListaAcotada.java`
  - `src/Tests/TestsContacto.java`
  - `src/quepasa/Contacto.java`
- Al final del documento tienes un poco de ayuda sobre compilación y ejecución.

## Listas Acotadas

**Ejercicio 1.** Nuestro equipo de desarrollo necesita una implementación de *listas acotadas*. Para ello ha diseñado un *tipo abstracto de datos* en forma de *interface* en Java. Dicho tipo, además de su nombre `List`, tiene las siguientes operaciones: `add`, `get`, `indexOf`, `remove`, `removeElementAt`, `set` y `size`. Su **semántica**:

- **`void`** `add(`**`int`** `insertIndex, E element)`: Coloca un nuevo elemento `element` en la posición `insertIndex` de la lista.
- `E get(`**`int`** `getIndex)`: Devuelve el elemento de la lista en la posición `getIndex`.
- **`int`** `indexOf(E search)`: Devuelve la posición ocupada por el primer elemento de lista igual a `search` (se usa `equals` para hacer la comparación). Devuelve `-1` si no se encuentra.
- **`boolean`** `remove(E element)`: Elimina de la lista el primer elemento que sea igual a `element` (se usa `equals` para hacer la comparación).
- **`void`** `removeElementAt(`**`int`** `removalIndex)`: Elimina de la lista el elemento que ocupa la posición `removalIndex`.
- **`void`** `set(`**`int`** `insertIndex, E element)`: Coloca el elemento `element` en la posición `insertIndex` (sobreescribiendo el elemento que ocupara dicha posición).
- **`int`** `size()`: Devuelve el número de elementos en la lista.
- **`int`** `isFull()`: Indica si la lista está llena.

El equipo ha colocado el interface en un paquete: `acotados.Lista` y te lo ha dejado en el directorio `src/acotados/Lista.java` del código de apoyo.

**Se pide:** desarollar la clase `acotados.ListaAcotada` que *implemente* dicho *interface* con un constructor que indica la capacidad máxima de la lista.

**Ejercicio 2.** En clase, Herranz ha mencionado el término *TDD*: *Test Driven Development* o *Desarrollo Dirigido por los Tests*. Tu equipo quiere que apliques esta técnica **antes** de desarrollar la clase `acotados.ListaAcotada`. Para ello ya sabes que tienes que empezar con una implementación *vacía*, es decir, que simplemente compile, de la clase `acotados.ListaAcotada`. Con esa implementación, y sólo para poder compilar, tendrás que desarrollar una serie de tests que ejecuten todas las operaciones de tu implementación.

**Se pide:** desarollar un programa principal `TestsListaAcotada` con un conjunto de tests (en forma de aserciones `assert`) sobre la clase `acotados.ListaAcotada`.

## Una nueva App

**Ejercicio 3.** Hasta ahora tu equipo no quería revelarte la verdadera razón de las peticiones anteriores. Lo que están haciendo es la nueva *App* con la que vais a destronar a *WhatsApp*. La nueva petición del equipo es que desarrolles la clase `quepasa.Contacto`. Dicha clase será la encargada de representar un contacto en la nueva *App*. Esta vez lo van a dejar todo en tus manos para que elijas incluso el nombre de los métodos. Las directrices son:

- Los contactos tienen un nombre (ej. `"John Doe"`).
- Los contactos tienen un avatar en forma de URL. (ej. `"https://pbs.twimg.com/profile_images/682327338755854336/gl1v_Qf6_400x400.jpg"`).
- A un contacto se le puede *enviar* un mensaje que queda almacenado en una lista interna de mensajes (ej. `"Hola guapo, quedamos para cenar?"`).
- En cualquier momento se le pueden pedir todos los mensajes enviados al contacto (los mensajes son simplemente `String`s).
- En cualquier momento se le puede pedir el último mensaje enviado.
- En cualquier momento se le pueden pedir el número de mensajes que se han enviado al contacto (un entero).

**Se pide:** aplicar *TDD* e implementar un programar principal `TestsContacto` con tests (en forma de aserciones `assert`) sobre la clase `quepasa.Contacto` que tienes pensado implementar.

**Ejercicio 4.** Tras haber escrito los tests `TestsContacto`, ahora ya puedes implementar la clase `quepasa.Contacto`.

**Se pide:** implementar la clase `quepasa.Contacto`.

## Ayuda

A modo de ayuda te cuento cómo puedes compilar y ejecutar desde la línea de comandos. Asumo que has descomprimido el *.zip* en un directorio y que abres una consola y te cambias hasta dicho directorio[1].

- **Compilar**[2]

```
$ javac -d bin -cp src -s src src/TestsListaAcotada.java
$ javac -d bin -cp src -s src src/TestsContacto.java
```

- **Ejecutar** (con *aserciones* habilitadas: `-ea`):

```
$ java -ea -cp bin TestsListaAcotada
$ java -ea -cp bin TestsContacto
```

[1] Los comandos han sido ejecutados en Unix, si usas Windows tendrás que utilizar "\" en vez de "/" en los nombres de los ficheros

[2] `javac` ya sabe compilar todo lo que depende del fichero indicado