Toxi-taxi Wicket - ejercicios

El objetivo de esta serie de ejercicios es acompañar el aprendizaje del desarrollo de aplicaciones Web usando Apache Wicket, mediante el agregado de funcionalidad a la interfaz Web implementada para el proyecto Toxi-taxi. Eventualmente, convendrá también hacer cambios al modelo.

Ejercicio 1

Agregar a la página que muestra los datos de un pasajero, la dirección principal.
Tener en cuenta qué hay que agregar en PasajeroPage.html y en PasajeroPage.java.

Atención: si en PasajeroPage.java agregamos un Label con este código

```
    this.add(new Label(
        "direccion",
        new PropertyModel<>(this.controller, "pasajero.direccionPrincipal")
    ));
```

va a mostrar un identificador horrible, tipo

```
   ar.unq.edu.cpi.toxitaxi.Direccion@3edfa6b3
```

porque lo que devuelve la expresión entre comillas, `"pasajero.direccionPrincipal"`, es un objeto Direccion. Conviene llegar a un método que devuelva un String. Buscar algún método conveniente en la clase Direccion, en lo posible que logre que se muestren calle y número.

Ejercicio 2

Agregar a la página que muestra los datos de un pasajero, la cantidad total de viajes, y si tiene o no viajes pendientes.

Atención:
el modelo no provee una forma sencilla de saber si un pasajero tienoe o no viajes pendientes. Este caso refleja una situación común en el desarrollo de software: al armar una interfaz que muestra un modelo,encontramos que el modelo no provee la funcionalidad que requiere la interfaz.
Cuando esto ocurre, debemos decidir si modificamos el modelo, o bien incluimos la lógica que haga falta dentro de la interfaz.

En este caso, la recomendación es ampliar el modelo para que nos sepa decir, dado un pasajero, si tiene o no viajes pendientes.
Esto nos lleva a otra decisión: a qué objeto agregarle la responsabilidad de brindar esta información.
Lo más natural es preguntarle al pasajero. Pero ooops ... los viajes los está manejando el CallCenter.

La elección de la opción más conveniente puede depender de las prácticas que se definan para cada proyecto, en particular cómo se manejan los datos.
Para este caso, digamos que vale agregarle la responsabilidad al Pasajero.
Si agregamos en la clase Pasajero un método de esta forma

```
    public List<Viaje> getViajes() {
        return CallCenter.unico().getViajesDe(this);
    }
```

entonces podemos preguntarle los viajes del pasajero al mismo pasajero, aunque la información la maneje el CallCenter.

Moraleja importante:
no es necesario que el objeto al que le preguntamos algo sea el mismo que responde.

Nota adicional:
Vale también agregar un método para obtener la cantidad de viajes.

Ejercicio 3

Agregar un pasajero al CallCenter, y poner un botón más en la pantalla.

Atención sobre esto:
para evitar conflictos con eventuales cambios que querramos en el CallCenter, definimos una clase aparte llamada DatosAgregados, para que cada uno agregue los datos que necesite. Modificar esa clase, agregando lo que haga falta en el método agregarPasajerosAdicionales().

Ejercicio 3.bis ¡¡desafío!!

Reemplazar los (ahora) cuatro botones fijos por una cantidad dinámica de botones o links, uno por cada pasajero que esté registrado en el CallCenter.

Ejercicio 4

Agregar una página que muestre los datos de un chofer, con tres botones correspondientes a cada uno de los choferes que están cargados en el Call Center.
Para cada chofer, mostrar: nombre, teléfono, e importe total cobrado.

Nota:
Se agregaron al modelo los pagos que recibe cada chofer. Hay algunos pagos cargados, para eso ver la clase EmpresaDeRemises. Se pueden cargar más pagos en la clase DatosAgregados, modificando el método cargarPagosAChoferAdicionales().

Dónde implementar la nueva página:
sse crearon la clase y el HTML llamados `DriverPage`, que están en el package `ar.unq.edu.cpi.toxitaxi.ui.driver1`. Esta página es invocada desde el menú inicial en la opción llamada "Choferes y pagos - versión 1".
Se pide que hagan la implementación de la nueva página completando `DriverPage` y agregando las clases que hagan falta en el mismo package. De esta forma se evita tocar el menú inicial, y mantener separado lo que implementa cada uno del punto de partida común.

Ayuda para armar la nueva página:
se recomienda pensar en los siguientes aspectos, eventualmente tomando notas con papel y lápiz.

- ¿Qué tags del HTML van a necesitar un `wicket:id`?
  Recordar que estos son los que van a tener que estar agregados en la clase Java de la página.
- ¿Qué información va a tener que manejar el controller?
- ¿Qué eventos hay que manejar, qué hay que hacer ante cada uno?

Ejercicio 5

Agregar al menos dos viajes adicionales, modificando el método `agregarViajesAdicionales()` en la clase DatosAgregados. Verificar que los viajes agregados se ven en la tabla de viajes del pasajero de cada uno.

Nota:
para hacer evolucionar a un viaje, usar estos métodos:

- `seRecibioPedido`, deja al viaje pendiente.
- `seAsignoViaje`, el viaje pasa de pendiente a en viaje.
- `seTerminoViaje`, el viaje pasa de en viaje a terminado.
- `seDescartoViaje`, el viaje pasa de pendiente a descartado.

Revisar el enunciado del ejercicio ToxiTaxi para más detalles sobre el ciclo de vida de un viaje.

Ejercicio 6

Agregar a la tabla de viajes definida en `ViajesPasajeroPage`, una columna que muestre el importe de cada viaje.

Ejercicio 7

Crear una página donde se muestren los pagos hechos a un chofer. Para cada pago: fecha, importe y nombre de la agencia. Usar una tabla. Incluir en la página un encabezado con nombre y teléfono del chofer.
Agregar un link a la página que muestra los datos de un chofer, para que redireccione a la página de pagos de ese chofer.

Consejo: el método getPagosRecibidos() en Chofer les va a servir.

Ejercicio 8

En la página creada en el ejercicio anterior, agregar abajo de la tabla de pagos un recuadro para poner el nombre y la dirección de una agencia. Se sugiere esta forma

Agencia:	**Dirección**:

Los datos de la agencia quedan vacíos al principio.

Transformar el dato de nombre de la agencia en un link. Cuando se pulsa, completar los datos de la agencia en el recuadro con la agencia elegida.

Sugerencia:
pensar qué cosa adicional debe manejar el controller de la página de pagos de un chofer.

Mejoras:

- Que el recuadro aparezca solamente cuando se elige una agencia.
- Que el recuadro aparezca a la derecha de la tabla, en lugar de abajo.
- Pensar alguna otra forma en la cual mostrar la dirección de la agencia que se elija.

Ejercicio 9

Agregar la posibilidad de cargar un nuevo pago a chofer, que puede invocarse desde un botón en la página de pagos a un chofer. Hay que pedirle al usuario: en qué agencia se hizo el pago, y el importe pagado. Como fecha de pago tomar la del día, como chofer el de la página de pagos desde la que se lanza la carga del nuevo pago.

Atención:
el constructor de la clase Java necesita un parámetro, que es el chofer. No conviene que compartan el controller entre las páginas, con pasarse el chofer alcanza. Otra opción es que la página de carga de pago conozca dos controllers: el suyo propio, y el de la página desde la que se invocó.