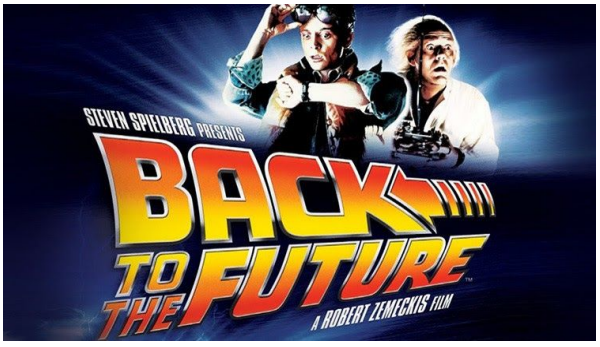
Volver al Futuro



En esta ocasión el Dr. Emmett Brown nos contrató para realizar un sistema que lo ayude a mantener un registro de sus viajes en el tiempo. Como el doctor es conocido de algunos docentes de UTN, pide que el sistema sea hecho en objetos.

Tenemos varios personajes que viajan en el tiempo. El objetivo principal es ver qué les va pasando a medida que avanza (?) el tiempo

Personajes

De cada personaje conocemos características personales y elementos que llevan consigo. Puede haber muchos elementos diferentes y de todos conocemos una descripción y su fecha, pero en especial es importante la documentación donde pueden aparecer los personajes de las película, como ser fotos o periódicos. También se destacan algunos elementos que fuera de época pueden provocar situaciones extrañas, como por ejemplo un libro de resultados deportivos, de los cuales se registra quién es el dueño original.

Algunos de los personajes, por ejemplo, son:

- Dr Emmett Brown (conocido como "Doc"): Tiene 71 años y mide 1,80. Lleva lentes de 1985, un control remoto fabricado en 1985, un perro llamado einstein de 1980, una foto de 1885 donde está él con un reloj, y un recorte de una revista de la UTN de diciembre de 2016 donde no aparece ningún personaje de la película.
- Marty Mcfly: Tiene 25 años y mide 1,70. Lleva una patineta voladora del 2015 de la que es el dueño y una foto de 1985 donde está él y sus hermanos.
- Jennifer Parker: Tiene 26 años y mide 1,50. En este momento no lleva nada.

Se pide:
Modelar los personajes y permitir averiguar:

1. Si son mayores (Más de 50 años) . Método **esMayor( )**
2. Si tiene algún elemento propio, por ejemplo una foto o cualquier otro documento en la que aparece o un elemento del que es dueño. Método **tieneElementoPropio( )**

Viajes

Para viajar, los personajes deben estar subidos al *DeLorean*, activar el condensador de flujo, indicar la fecha de destino en el panel digital, y acelerar hasta llegar a la velocidad de 88 mph.
Los viajes se realizan en el vehículo favorito de todos, el *DeLorean*, que usa barras de plutonio como combustible, pero en ocasiones se puede reemplazar por otro y suceden cosas diferentes.

- Al utilizar plutonio o cualquier elemento radiactivo, es casi imposible evitar los efectos secundarios y la altura de todos los viajeros se reduce en 1 cm.
- Cuando usan nafta, el combustible habitual del DeLorean, los personajes mayores rejuvenecen 10 años de edad, mientras que los menores envejecen 5.

- Si se utiliza energía eléctrica, por ejemplo de un rayo que cae sobre el reloj del pueblo, a todos los viajeros se les pierde la cosa más antigua que llevaban consigo. (en caso que no llevaran nada, se produce un error).
- También, otra posibilidad es utilizar basura, lo cual genera fuertes olores hacen al viaje en el tiempo una tarea difícil de sobrellevar que de alguna manera afecta a los viajeros (inventar una forma sencilla pero no trivial en que sean afectado los viajeros).

Se pide:

1. Implementar el método **viajarA(unDestino,unaFecha)** que representa un viaje en el tiempo en el *DeLorean* al destino indicado, en el que sucede lo que corresponda. No es necesario llevar un registro de viajes realizados.
2. Mediante otro métodos, permitir cambiar el combustible, subir y bajar personajes del vehículo.
3. Hacer tests de al menos dos viajes

Situaciones especiales

Los antepasados

Uno de los viajes, en el que van Doc y Marty en el DeLorean, tiene por destino el lejano oeste en el siglo XIX. Allí se encuentran con muchos personajes, entre quienes hay familiares de Marty.

- Se quiere averiguar quienes son todos los personajes del destino, que tengan el mismo apellido que un personaje indicado y que sean mayores. Implementar el método **antepasadosDe(unPersonaje).**

Fotos que se desvanecen

Cuando alguno de los viajeros viaja al pasado e interviene en la realidad del momento, puede generar cambios con consecuencias impensadas. Cuando eso sucede, las fotos y otros documentos del futuro se desvanecen o modifican. Por ejemplo, en una oportunidad, Marty altera la historia de sus padres y hace que tanto él como sus hermanos desaparezcan de la foto que llevaba.

- Para los ocupantes actuales del DeLorean, dado un personaje y una fecha, hacer que dicho personaje desaparezca de todas las fotos u otros documentos futuras que tengan los viajeros y que todo elemento que sea del futuro agregue a su descripción el prefijo "BTTF". Implementar el método **huboUnProblemaCon(unPersonaje,unaFecha)**

Paradojas temporales

Una de las paradojas que puede suceder es que el viajero se encuentre consigo mismo en un futuro o pasado. Por ejemplo, Jennifer se encuentra consigo misma en uno de los viajes. En la película, son la misma persona, pero en dos momentos diferentes de su vida, con distinta edad o altura. ¿Cómo se podría modelar en el paradigma de objetos? No implementarlo, simplemente explicarlo desde los conceptos.

Correcciones comunes

1) Usar polimorfismo entre documentos y otros elementos

en vez de discriminar a los documentos del resto

```
var aux = listaDeElementos.filter({elem=>elem.esDocumento()})
aux.forEach({documento=>documento.sufrirEfectoDeViajar(unPersonaje)})
```

a todos decirle directamente

```
listaDeElementos.forEach({documento=>documento.sufrirEfectoDeViajar(unPersonaje)})
```

y sufrirEfectoDeViajar(unPersonaje) en los que no son documentos que haga otra cosa.

2) Usar los métodos adecuados de colecciones

a)
en vez de hacer
```
deLorean.personajesAbordo().map( p => p.algo() … )
```
(donde algo() modifica a cada personaje y no se retorna nada)

utilizar foreach
```
deLorean.personajesAbordo().foreach( p => p.algo() … )
```

b)
Para eliminar el mínimo de una colección, en vez de ordenar y quitar el primero

```
method ordenarElementosPorAntiguedad() {
        elementos.sortBy({elem1,elem2=>elem1.getFecha()<elem2.getFecha()})
}
method perderLoMasAntiguo(){
        self.ordenarElementosPorAntiguedad()
        elementos.drop(1)
}
```

es mejor usar

```
method elementoMasAntiguo(){
        return elementos.min{e => e.fecha()}
 }
 method perderLoMasAntiguo(){
        elementos.remove{self.elementoMasAntiguo()}
 }
```

c)
Para ver si al menos uno cumple una condición
no usar filter

elementosQueLleva.filter({elem=> elem.condicion()}).size() >= 1

sino, más declarativamente, usar any()
elementosQueLleva.any({elem=> elem.condicion()})

3) Usar los objetos mismos, no uno de sus atributos con nombre, id o similar
aca, el objeto marty tiene un atributo, su nombre, con el string "marty".
var marty = new Personaje("marty","McFly",25,170,[foto, elem1, elem2])
en la foto, para indicar quienes estan, se ponen sus nombres, en este caso, el string "marty"
var foto= new Documentacion("Foto de Familia",1985, ["marty"])

en vez de esto, se debería utilizar el objeto marty directamente (sin comillas).
var foto= new Documentacion("Foto de Familia",1985, [marty])

El mismo problema se ve en
foto.personajes().contains(unPersonaje.apellido())

mejor
foto.personajes().contains(unPersonaje)

4) Evitar repetir lógica.
Si siempre los combustibles afectan a todos los pasajeros,
en vez de hacer el foreach en cada combustible

```
object auto {
...
combustible.efecto(personajesAbordo)
...
object radioactivo {
	method efecto(listaDePersonajes) {
		listaDePersonajes.forEach({personaje=> personaje.modificarAltura(-1)})
	}
}
```

o tambien

```
object auto {
...
combustible.efecto(self)
...
object radioactivo {
	method efecto(auto) {
		auto.listaDePersonajes().forEach({personaje=> personaje.modificarAltura(-1)})
	}
}
```

mejor hacer un unico foreach del lado del auto, y pasarle al combustibles los personajes de a uno

```
object auto {
...
listaDePersonajes().forEach({personaje=> combustible.efecto(personaje) })
...
object radioactivo {
	method efecto(personaje) {
		personaje.modificarAltura(-1)
	}
}
```

5)
Clasificar como objetos diferentes a las fotos, elementos especiales y otros, no con atributos con strings.

```
method fotos(){
	return elementos.filter({elem => elem.descripcion() == "foto" )
}
```

tampoco está bueno hacer tipo bandera

```
class Elemento
method foto() = false

class Foto
method foto() = true
```

y despues preguntar

```
if( algo.foto())
```

o

```
elementos.filter{ elemento => elemento.foto()}
```

6) Otro polimorfismo no aprovechado

Se puede ver en

```
method tieneElementoPropio() {
	return elementos.any{ elemento => elemento.duenio() == self}
	|| elementos.any{ elemento => elemento.aparecePersonaje(self)}
}
```

esta definido los metodso duenio y aparecePersonaje en todos los elementos, o solo en los documentos y especiales? que retorna duenio en un elemento comun?

Lo correcto es preguntar directamente lo que se quiere saber, si algun elemento es propio.

```
method tieneElementoPropio() {
	return elementos.any{ elemento => elemento.esDe( self ) }
}

 class Elemento{
	...
	method esDe(alguien) = false
 }
```

```
class Documentacion inherits Elemento{
	var personajes
	override method esDe(alguien){
		return personajes.contains(alguien)
	}
}

class Especial inherits Elemento{
	const duenio
	override method esDe(alguien){
		return duenio == alguien
	}
}
```

7) No usar herencia para instanciar

en vez de:

```
object lentes inherits Elemento(1985, "Lentes")
```

hacer en donde se vaya a usar

```
var lentes = new Elemento(1985, "Lentes")
```

8) No usar herencia cuando no hay nada que heredar

```
class Combustible {
	method aplicarEfecto(pasajero)
}

class CombustibleRadiactivo inherits Combustible {
	override method aplicarEfecto(pasajero) {
		pasajero.disminuirAltura( 1 )
	}
}
```

Directamente

```
class CombustibleRadiactivo {
	method aplicarEfecto(pasajero) {
		pasajero.disminuirAltura( 1 )
	}
}
```

o incluso, con un objeto bien conocido, sin necesidad de clase

9) Delegar adecuadamente, no repetir lógica

Por ejemplo, varios errores confluyen en

```
object cualquiera {
```

```
method huboUnProblemaCon(unPersonaje,unaFecha){
	deLorean.pasajeros().forEach({x => self.borrarDeFotos(unPersonaje,x.elementos())})
	deLorean.pasajeros().forEach({x => self.agregarPrefijo(x.elementos(),unaFecha)})
	}
```

Los metodos auxiliares son:

```
	method borrarDeFotos(personaje,elementos){
		elementos.forEach({x => self.borrar(personaje,x)})
	}
	method borrar(personaje,elemento){
		if(elemento.esFoto() && elemento.aparece(personaje))
			elemento.borrarPersona(personaje)
	}
	method agregarPrefijo(elementos,fecha){
		elementos.forEach({x=>self.elementoDelFuturo(x,fecha)})
	}
	method elementoDelFuturo(elemento,fecha){
		if(elemento.esDelFuturo(fecha))
			elemento.setDescripcion("BTTF" + elemento.getDescripcion())
	}
}
```

Debería ser responsabilidad del objeto deLorean hacer una tarea con sus pasajeros.
Si cada pasajero debe hacer una tarea y a continuación cada pasajero debe hacer otra, es mas facil delegar de manera que cada pasajero haga dos tareas juntas.
Es mejor perdirle directamente al pasajero que haga la tarea con los elementos que lleva en vez de pedírselo a un objeto cualquiera y pasarle los elementos por parámetro.
Más aún, el pasajero debería delegarle a su vez a cada elemento para que haga la tarea concreta.
Además, nuevamente aparecen los problemas de manejar mal el polimorfismo de las fotos

Quedaria asi

```
object deLorean {
method huboUnProblemaCon( unPersonaje, unaFecha)
	pasajeros.forEach({pasajero => pasajero.problemaCon( unPersonaje,  unaFecha)    }

class Pasajero {
	…
	method problemaCon( unPersonaje, unaFecha){
		elementos.filter{e => e.esMasReciente( unaFecha )}
			.forEach{e => e.problemaCon( unPersonaje )}
		}
```

y recién en el método problemaCon() de cada tipo de elemento se resuelve si se borra de la foto y agrega el prefijo, desde lo que pide la consigna, no se está manejando la fecha bien.

```
class Elemento{
	var descripcion
	…
	method problemaCon(unPersonaje){
		descripcion = "BTTF " + descripcion
	}
```

```
}

class Documentacion inherits Elemento{
	var personajes
	…
	override method problemaCon( unPersonaje ){
		super(unPersonaje)
		 if(self.esDe(alguien))
			personajes.remove(alguien)

	}
}
```

10)

no ser redundantes en las condiciones ni abusar del if
en vez de

```
if (edad >50){
	return true
} else {
	return false
}
```

hacer directamente

```
return edad > 50
```

11) Otro caso de polimorfismo

en vez de

```
method viajarA(destino,fecha){
		if (combustible==radiactivo) {
			personajes.forEach({persona => persona.cambiaraltura()})
		}
		if (combustible==nafta){
			personajes.forEach({persona => persona.cambiaredad()})
		}
		if (combustible == energiaElectrica){
			personajes.forEach( {p => p.eliminarElementoMin()} )
		}
		if (combustible==basura){
			personajes.forEach( {p => p.eliminarElementoMax()} )
		}
	}
```

Usar polimofirsmo entre los objetos de los combustibles y delegarles que hagan algo sobre cada pasajero

```
method viajarA(destino,fecha){
	personajes.forEach({persona => combustible.cambiar(persona)})
```

```
}
```

Dejado fuera NO HACER

Además el delorean no es el único vehículo temporal existente, Marty McFly ha vuelto del futuro con un Ferrari temporal. Debido a las fuertes velocidades y al espacio reducido de los mismos, las personas altas no pueden viajar en él… Pero las bajas si, y cuando lo hacen su nivel de Felicidad se incrementa en 100 puntos!

3- Evitar, de la forma que se crea más conveniente, que una persona alta pueda viajar en un Ferrari, no queremos dañar su salud!

El Dr. necesita poder defenderse de los libios que quieren atacarlo por el plutonio que se ha robado. Para eso necesita armas, de todo tipo, de cuchillos hasta bazookas.. Todas las armas afectan de manera similar a la persona que la usa, aumentan su nivel de coraje en la cantidad de puntos de potencia que tenga el arma. Armas hay de todos los estilos, punzantes (cuchillos, estrellas y demás) que reducen en 50 los puntos de vida del libio. Las armas de fuego matan al libio (dejan sus puntos de vida en 0) y las armas explosivas (Bazookas y granadas) que hacen que el libio se arrepienta y le entregue más plutonio al Dr. (tanto como el libio disponga).
No es necesario implementar. ¿Qué cambios cree conveniente realizar? Justifique.

Todo el mundo es capaz de viajar en el tiempo, pero no a todos les afecta de la misma forma. Para la gente mayor de edad, como el Dr. Emmett Brown, cada viaje le provoca un síntoma diferente, estos pueden ser, dolor de espalda, un resfriado o alguna idea loca.
La gente joven (de menos de 60 años) siempre se pone mareada al viajar, excepto la familia McFly, ellos no tienen síntomas cuando viajan.
Los síntomas de la gente mayor (de mas de 60 años), se manifiestan de forma diferente dependiendo del auto en el que viajan y del estado del mismo. El auto puede ser un Delorean que puede funcionar con radiación, basura o nafta. Al utilizar basura como combustible, siempre se pescan un resfriado y al utilizar radiación, alguna idea loca se les viene a la mente.
Además el delorean no es el único vehículo temporal existente, Marty McFly ha vuelto del futuro con un Ferrari temporal. La gente mayor al viajar en el ferrari se queda con un fuerte dolor de espalda.