Apellidos:	**Nombre:**
UO:	**Firma:**

Se dispone del código fuente de un prototipo de videojuego basado en la franquicia Pokemon. Tal y como se puede ver en la figura 1, en este juego se establecen combates entre las criaturas (clase **Pokemon**) de diferentes entrenadores (clase **EntrenadorPokemon**). Cada entrenador dispone de un equipo formado por 2 pokemons. En el prototipo actual los equipos están formados siempre por un **Pikachu** y un **Charmander**. En el anexo 1 se pueden ver el código de estas clases.

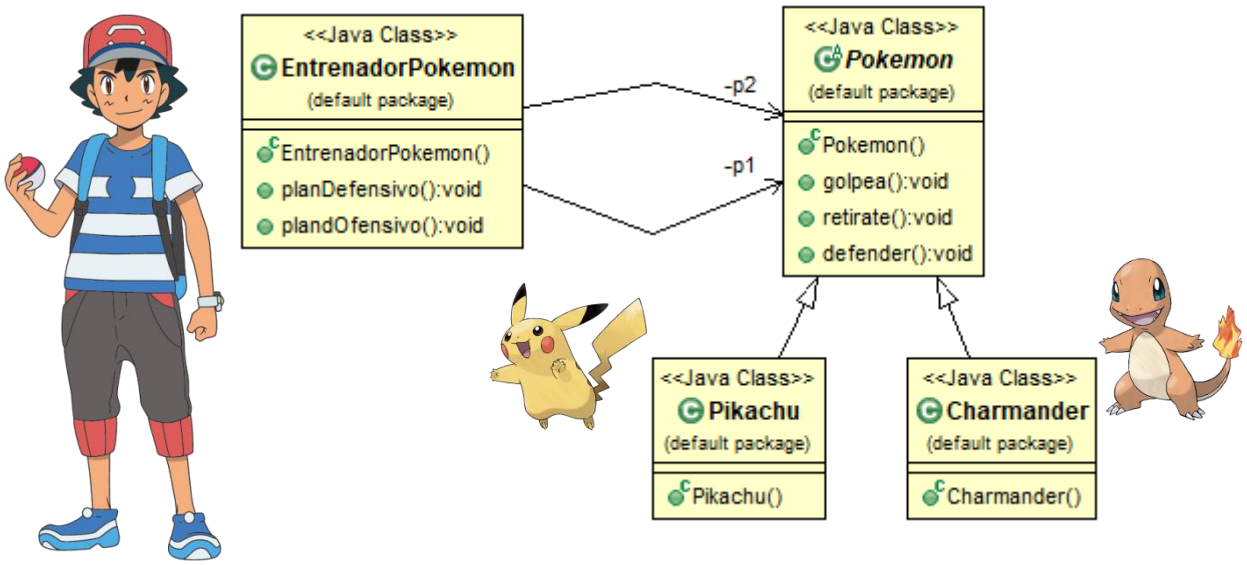

Figura 1: Clases disponibles en el prototipo inicial.

En el juego final pueden participar en realidad cientos de Pokemons distintos, cada uno con su propia subclase. Además se prevé que las órdenes que reciban los pokemons sean independientes de su tipo concreto.

Como responsables del software, se nos solicita que cualquier método del juego que necesite crear Pokemons debería estar diseñado de forma que su implementación sea independiente de los tipos de Pokemons disponibles en cada versión del juego. Así por ejemplo, cada nueva instancia **EntrenadorPokemon** podrá recibir en su constructor los dos **Strings** que indican el tipo de los pokemons que desea crear para formar con ellos su equipo. Así para crear un entrenador con un equipo como el del anexo 1 bastaría con:

EntrenadorPokemon ash = new EntrenadorPokemon(“Pikachu”, “Charmander”);

O por ejemplo si creamos las subclases de **Pokemon: Goldeen y Staryu** podríamos definir otro equipo:

EntrenadorPokemon misty = new EntrenadorPokemon(“Goldeen”, “Staryu”);

1.- Selecciona el patrón más adecuado para esta tarea. **Justifica tu respuesta**. Indica su tipo, y si el patrón utilizado tiene varias versiones justifica cual utilizarías. **[1,5 puntos]**

2.- Aplica el patrón seleccionado del punto 1. Describe qué función tiene cada nueva Interfaz/Clase nueva, e implementa las operaciones de cada una de ellas. Actualiza la definición de la clase **EntrenadorPokemon** para generar ese mismo equipo inicial por defecto: {Pikachu, Charmander} **[2,5 puntos]**

Se nos ofrece la posibilidad de incorporar las criaturas de otra franquicia a nuestro videojuego: Digimon. La jerarquía de criaturas de Digimon sigue el diagrama de la figura 2. Como podemos ver por cada Digimon existe una subclase de **Digimon**. En el diagrama solo aparecen 2 de las decenas de digimons que existen. Además nos indican que nos permiten utilizar las clases Digimon, pero **no modificarlas** (de ahí el icono del candado).

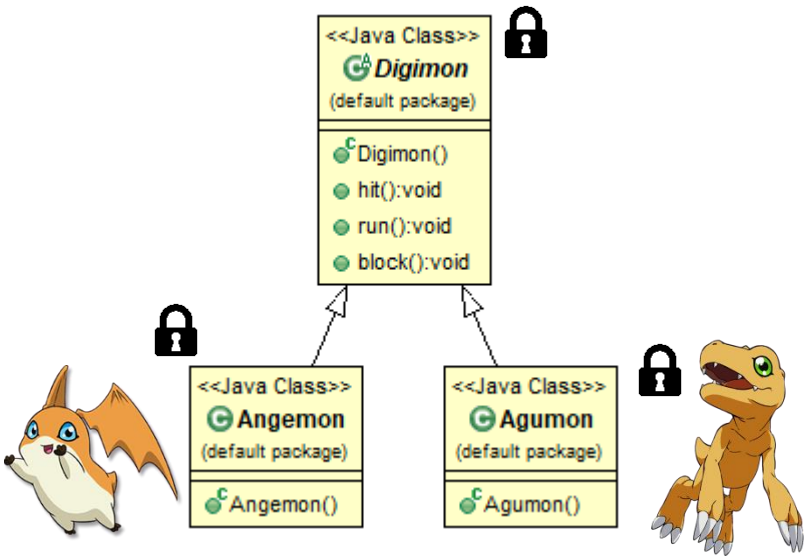

Figura2: Jerarquía de clases de las criaturas **Digimon**. **Angemon** y **Augumon** son solo un par de las decenas de Digimons existentes en la franquicia.

Se nos solicita aplicar el patrón más adecuado para poder utilizar en nuestro videojuego tanto criaturas Pokemon como Digimon. Así, todos los Digimons deben poder ser manipulados como si fueran nuevos tipos de Pokemons. Además se requiere que, la incorporación de cada nuevo tipo de Digimon **no suponga la creación de nuevas clases**, sino que la solución aportada permita utilizar como **Pokemons** a cualquier futura criatura que herede directa o indirectamente de la clase **Digimon**.

3.- Selecciona el patrón más adecuado para esta tarea. **Justifica tu respuesta**. Indica su tipo, y si el patrón utilizado tiene varias versiones justifica cual utilizarías. **[1,5 puntos]**

4.- Aplica el patrón seleccionado del punto 3. Describe qué función tiene cada nueva Interfaz/Clase nueva, e implementa las operaciones de cada una de ellas. Actualiza la definición de la clase **EntrenadorPokemon** para generar el siguiente equipo inicial de "pokemons": un **Angemon** y un **Agumon [3 puntos]**

A la vista de los métodos **planDefensivo**() y **planOfensivo**() de la clase **EntrenadorPokemon** los autores del juego se dan cuenta que para modificar los comportamientos de los pokemons es necesario reescribir una y otra vez estos métodos y volver a compilar el programa. Nuestro equipo de programadores se plantea que patrón de diseño nos puede facilitar el modificar y/o reprogramar los comportamientos de los Pokemons de un equipo sin tener que editar el código fuente y recompilar cada vez. Es decir que en tiempo de ejecución se pudiera (incluso por parte del usuario a través de la interface de la aplicación) programar los comportamientos de los pokemons: indicando que secuencia de acciones realizar y que pokemon debe realizarlas.

5.- Selecciona el patrón más adecuado para esta tarea. **Justifica tu respuesta**. Indica su tipo, y si el patrón utilizado tiene varias versiones justifica cual utilizarías. **[1,5 puntos]**

Anexo I: Código Fuente

```
public abstract class Pokemon {
	public void golpea(){
		// golpea a algún pokemon enemigo a tiro
	}
	public void retirate(){
		// vuelve a su pokeball
	}
	public void defender(){
		// el pokemon se concentra para recibir menos daño en el siguiente ataque
	}
}
```

```
public class Pikachu extends Pokemon {
	// ...
}
```

```
public class Charmander extends Pokemon {
	// ...
}
```

```
public class EntrenadorPokemon {
	private Pokemon p1; // primer pokemon de su equipo
	private Pokemon p2; // segundo pokemon de su equipo


	public EntrenadorPokemon(){
		p1 = new Pikachu();
		p2 = new Charmander();
	}

	public void planDefensivo(){
		p1.defender();
		p2.defender();
	}

	public void plandOfensivo(){
		p1.golpea();
		p2.golpea();
		p1.golpea();
		p2.golpea();
	}
}
```

```
public abstract class Digimon {
	public void hit(){
		// ataca a su enemigo (equivalente a Pokemon.golpea() )
	}
	public void run(){
		// huye del combate (equivalente a Pokemon.retirarse() )
	}
	public void block(){
		// bloquea el siguiente ataque de su enemigo (equivalente a Pokemon.defender() )
	}
}
```