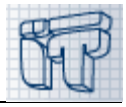


NOMBRE:
APELLIDOS:
NIA:
GRUPO:

Primer parcial

2ª Parte: Problemas (7 puntos sobre 10)

Duración: 80 minutos
Puntuación máxima: 7 puntos
Fecha: 9 marzo 2018

Instrucciones para el examen:

- No se permite el uso de libros o apuntes, ni tener teléfonos móviles u otros dispositivos electrónicos encendidos. Incumplir cualquiera de estas normas puede ser motivo de expulsión inmediata del examen.
- Rellena tus datos personales antes de comenzar a realizar el examen.

Problema 1 (5 / 7 puntos)

La empresa ACME está desarrollando una versión online del Monopoly. Nos han pedido modelar las casillas del tablero, mediante las siguientes clases:

- Casilla:
  - Representa una casilla genérica del tablero.
  - Sólo almacena el nombre de la casilla.
- Propiedad:
  - Representa una casilla especial que puede ser comprada y en la que hay que pagar un alquiler si pertenece a otro jugador.
- Solar:
  - Representa un tipo de propiedad especial (una calle), con un color, en la que se pueden edificar casas y hoteles.
- Estación (no es necesario programarla):
  - Representa un tipo de propiedad especial, una estación, que no se puede edificar y cuyo alquiler depende del número de estaciones que posea el propietario.

Apartado 1 (1 punto)

Programa la clase Casilla, que:

- Sólo almacena el nombre de la casilla (String).
- Proporciona un único constructor que recibe el nombre de la casilla.

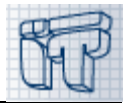
La clase Casilla permite instanciar objetos; cualquier casilla del tablero que no sea de tipo Propiedad (o uno de sus descendientes) se representará mediante un objeto de tipo Casilla.

Apartado 2 (2 puntos)

Programa la clase Propiedad, que:

- Es un tipo especial de Casilla.
- Proporciona un único constructor que recibe como parámetros:
  - Nombre de la casilla (String).
  - Importes del alquiler (array de int).

  E inicializa por defecto el propietario (String) a null.
- Los valores del propietario y alquileres deben ser accesibles desde las clases hijas.



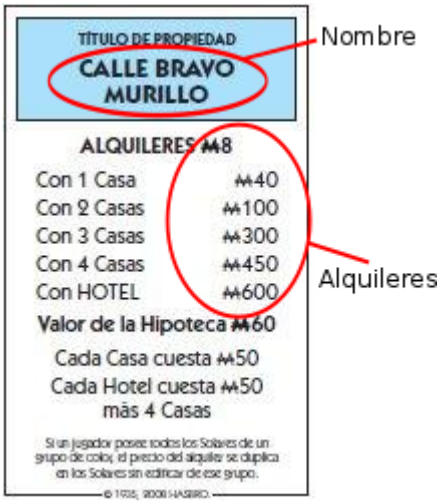
- Proporciona los métodos:
  - void comprar(String comprador)
    - Almacena *comprador* como propietario de la casilla
  - int getAlquiler()
    - Calcula el importe del alquiler a abonar. Ten en cuenta que este método no puede implementarse, porque el cálculo depende de si es una Estacion o un Solar.

Apartado 3 (2 puntos)

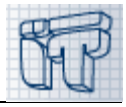
Programa la clase Solar, que:

- Es un tipo especial de Propiedad.
- Proporciona un único constructor que recibe como parámetros:
  - Nombre de la casilla (String)
  - Importes del alquiler (array de int, con los valores del alquiler para los casos en que no hay casas (posición 0), hay 1 casa, 2 casas, 3 casas y 4 casas (posición 4))
  - Color (String)

  E inicializa por defecto el propietario a null y el número de casas de la casilla a 0.
  Por simplicidad, no vamos a incluir hoteles en nuestra versión del juego.



- Proporciona los métodos:
  - comprarCasa()
    - Incrementa en uno el número de casas edificadas en la casilla.
    - Lanza una excepción de tipo MonopolyException (ya programada) si se intenta superar el máximo posible de casas (4).
  - int getAlquiler()
    - Devuelve el importe del alquiler
    - Si la casilla está libre (no tiene propietario), devuelve 0.
    - Si la casilla tiene propietario (es distinto de null), devuelve el importe almacenado en la casilla correspondiente del array de alquileres, dependiendo del número de casas edificadas.



Solución y Criterios Corrección:

//Aptdo 1: 1 punto

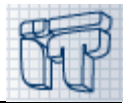
```
public class Casilla {

	private String nombre;
	public Casilla(String nombre) {
		this.nombre = nombre;
	}

}
```

// Aptdo. 2: 2 puntos

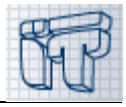
```
public abstract class Propiedad extends Casilla {

	protected int[] alquileres;
	protected String propietario;

	public Propiedad(String nombre, int[] alquileres) {
		super(nombre);
		this.alquileres = alquileres;
		propietario = null;
	}

	public void comprar(String comprador) {
		this.propietario = comprador;
	}

	public abstract int getAlquiler();

}
```

// Aptdo. 3: 2 puntos

```
public class Solar extends Propiedad {

	private String color;
	private int numCasas;

	public Solar(String nombre, int[] alquileres, String color) {
		super(nombre, alquileres);
		this.color = color;
		numCasas = 0;
	}

	public void comprarCasa() throws MonopolyException {
		if (numCasas >=4) {
			throw new MonopolyException("Máximo 4 casas");
		}
		numCasas++;
	}
	@Override
	public int getAlquiler() {
		int alquiler = 0;
		if (super.propietario != null) {
			alquiler = super.alquileres[numCasas];
		}
```



```
                return alquiler;
        }

}
```



Problema 2 (1 / 7 puntos)

La clase `MatrixTrilogyRepository` ofrece información sobre la trilogía de Matrix. Por simplicidad, se muestra a continuación únicamente el método `getReleaseYear` que devuelve el año del estreno de cada una de las películas de la trilogía en base a su posición en la trilogía.

```
public class MatrixTrilogyRepository extends FilmRepository {

  /**
  * Devuelve el año del estreno de cada una de las películas de la trilogía de Matrix
  * @param matrixTrilogyIndex Índice de la película dentro de la trilogía
  * @return Año de estreno de la película
  * @exception NotFoundException si el índice está fuera del rango de la trilogía
  */
  public int getReleaseYear(int matrixTrilogyIndex) throws NotFoundException {

       int releaseYear = -1;

       if (matrixTrilogyIndex == 1) {
              releaseYear = 1999;
       } else if ((matrixTrilogyIndex == 2) || (matrixTrilogyIndex == 3)) {
              releaseYear = 2003;
       } else {
              throw new NotFoundException("Unfortunately Matrix is a trilogy :(");
       }

       return releaseYear;
  }
}
```

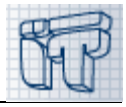
Apartado 1 (0,5 puntos)

Se pide identificar las clases de equivalencia.

Apartado 2 (1,5 puntos)

Se pide implementar la clase `MatrixTrilogyRepositoryTest` (JUnit test case) con los métodos de test que realicen una prueba de caja blanca con un grado de cobertura del 100% de líneas y ramas del método `getReleaseYear` de la clase `MatrixTrilogyRepository`.

Nota: el número de métodos de test que se implementen es libre siempre y cuando el caso de prueba cubra el 100% del método getReleaseYear.



Solución:

```
public class MatrixTrilogyRepositoryTest {

  @Test
  public void testGetReleaseYear() throws NotFoundException {
       MatrixTrilogyRepository matrixRepository = new MatrixTrilogyRepository();
       assertEquals(matrixRepository.getReleaseYear(1), 1999);
       assertEquals(matrixRepository.getReleaseYear(2), 2003);
       assertEquals(matrixRepository.getReleaseYear(3), 2003);
  }

  @Test(expected = NotFoundException.class)
  public void testGetReleaseYearNotFoundException( ) throws NotFoundException {
       MatrixTrilogyRepository matrixRepository = new MatrixTrilogyRepository();
       matrixRepository.getReleaseYear(5);
  }

}
```