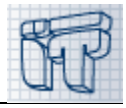


NOMBRE:
APELLIDOS:
NIA:
GRUPO:

Primer examen parcial

2ª Parte: Problemas (7 puntos sobre 10)

Duración: 90 minutos
Puntuación máxima: 7 puntos
Fecha: 15 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 puntos)

Te han contratado los dueños del restaurante "*Hungry Birds*" para mejorar la gestión del negocio. Para ello, decides implementar un programa Java que permita modelar las personas involucradas haciendo uso del menor código posible. Así, podrás almacenarlas en una base de datos y tener un registro. Por organización, has decidido partir de la interfaz `GestionRestaurante`, que debe ser implementada por las clases que programes en los primeros 3 apartados:

```
public interface GestionRestaurante {
    void tarea();
}
```

Apartado 1 (1 punto)

Programa la clase `Persona`, que representa a las personas en general y servirá como base para modelar después los trabajadores y los clientes. Cada persona tendrá un (1) nombre (`nombreUsuario`) y (2) género (`genero`). Este último decides guardarlo como un carácter que representa a hombres (`m`) y mujeres (`f`). Todos los atributos deben ser visibles únicamente desde esta clase. Además, decides programar:

- Un constructor, que recibe como argumentos el nombre del usuario y su género, y asigna el valor correspondiente a los atributos.
- Un método `getNombreUsuario()` que devuelve el nombre de usuario de la persona.
- Un método `toString()` que devuelve un String con los datos de la persona del siguiente modo.

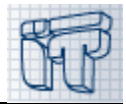
  `EvaGarcia_3 (f).`

- No se dispone de información suficiente para implementar el método `tarea()` de la interfaz.

Apartado 2 (1,5 puntos)

Programa la clase `Trabajador`, que representa a los trabajadores del restaurante. Cada trabajador, además de nombre de usuario y género, tendrá un `sueldo` y un `puesto` (sólo visibles en esta clase):

- Promoviendo la igualdad, los dueños del restaurante les pagan el mismo salario a todos los trabajadores. Así, el `sueldo` es común a todos los empleados. Este trimestre son 1600 euros, aunque no es fijo porque puede aumentar cada trimestre si hay grandes beneficios en el negocio.
- El `puesto` se modelará mediante constantes que pueden tomar los siguientes valores:
  - `MESA = "camarero"` `COCINA = "cocinero"`



El valor por defecto si el usuario introduce un valor no válido será `MESA`.

Además, decides programar:

- Un constructor que recibe, en este orden, el puesto, el nombre del usuario y su género, y asigna el valor correspondiente a los atributos. Tendrá que comprobar que el puesto es correcto y si no lo es, asignar el valor por defecto.
- El método `tarea()`, que muestre por pantalla.
    - Si el trabajador es un camarero

      `<nombreUsuario>, sirvo las mesas.`
    - Si el trabajador es un cocinero

      `<nombreUsuario>, preparo la comida.`

Apartado 3 (1,5 puntos)

Programa la clase `ClienteVIP`, que representa a los clientes habituales del restaurante, a los que se les entrega una tarjeta personal. Además de nombre de usuario y género, cada cliente VIP tiene una `tarjeta` asociada, cuya información tendrás que guardar. Para ello, decides hacer uso de la clase `Tarjeta`, que puedes suponer implementada y con un constructor que recibe tantos parámetros como atributos tiene. Los atributos de esa clase son el número de la tarjeta (`numero`) y la fecha de creación (`creacion`). En caso de que fuera necesario, puedes acceder a ellos con los métodos `getNumero()`, `setNumero(int num)`, `getCreacion()` y `setCreacion(String fecha)`.

Además, en tu clase `ClienteVIP` decides implementar:

- Un constructor que reciba el nombre de usuario y género de la persona, junto con el número de tarjeta y la fecha de creación, y asigne los valores correspondientes a los atributos.
- El método `toString()` que devuelva un `String` con la información disponible sobre el cliente del siguiente modo.

  `EvaGarcia_3 (f). Tarjeta número 12345 creada el 15/03/2018`
- El método `tarea()`, que muestre por pantalla.

  `Cliente VIP, vengo a comer aquí.`

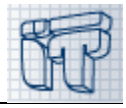
Apartado 4 (1 punto)

Programa la clase `TestRestaurante`, que será la clase principal de tu programa, en la que probarás tu código. Para ello incluye un método `main`. En ese método deberás inicializar un `ArrayList` en el que incluirás un cocinero y un cliente. Además, debes invocar el método `toString()` que has implementado en los apartados anteriores, sobre las entradas de tu `ArrayList` de forma que el resultado que se imprima por pantalla sea:

```
EvaGarcia_3 (f).
AndreaIzq (f). Tarjeta número 12345 creada el 15/03/18
```

NOTA 1: No debes preocuparte por importar la clase `ArrayList<E>`.
NOTA 2: La clase `ArrayList<E>` tiene los siguientes métodos, algunos de los cuales pueden ser de utilidad:

- `boolean add(E e)`
- `void add(int index, E element)`
- `void clear()`
- `E get(int index)`
- `int indexOf(Object o)`
- `boolean isEmpty()`
- `boolean remove(Object o)`
- `int size()`



Problema 2 (2 puntos)

En una empresa, tenían un cierto programa implementado, del cual tenían sus pruebas unitarias. Sin embargo, no tenían copia de seguridad y al dañarse su disco duro, han perdido el código original, aunque no el fichero de pruebas. Se sabe que el método se encargaba de sumar los dígitos de cadenas de texto. El código que prueba el programa (que está programado perfectamente) es el siguiente:

```
public class TestUnknown {

	@Test(expected=NumberFormatException.class)
	public void test() {
		assertEquals(Unknown.method("22"),4);
		assertEquals(Unknown.method("1234"),10);
		assertEquals(Unknown.method("11111"), 5);
		assertEquals(Unknown.method("00000"), 0);
		assertEquals(Unknown.method("abc"),-1);
	}
}
```

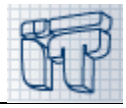
Apartado 1 (1 punto)

Se pide implementar el código original de la clase perdida. Asuma que la clase original únicamente tenía el método que aparece en el enunciado. De cara al tratamiento de excepciones, realice únicamente la mínima gestión que sea imprescindible. Como ayuda para la implementación del método, puede utilizar el método `static int parseInt(String s)` de la clase `Integer`.

Apartado 2 (1 punto)

En caso de que se incluyan nuevos casos en un futuro, se decide transformar el código de pruebas en un test paramétrico. Complete el siguiente código para transformar el código de pruebas que se muestra al inicio del enunciado sin perder funcionalidad. Para ello, puede usar el método `assertEquals`, cuyos parámetros se corresponden con los del ejemplo de la clase `TestUnknown`.

```
@RunWith(Parameterized.class)
public class TestUnknown2 {

	@Parameters
	public static Collection<Object[]> data(){
		Object[][] data = new Object[][]{_____________________________
		_____________________________________________________________
		_____________________________________________________________};
		return Arrays.asList(data);
	}

	@Parameter(1)
	public _____________ a;

	@Parameter(0)
	public _____________ b;

	@Test
	public void test(){









	}
}
```



SOLUCIONES DE REFERENCIA (Varias soluciones a cada uno de los problemas son posibles)

PROBLEMA 1

Apartado 1 (1 punto)

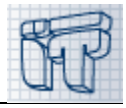
```
public abstract class Persona implements GestionRestaurante {

    private String nombreUsuario;
    private char genero;

    public Persona(String nombreUsuario, char genero){
        this.nombreUsuario = nombreUsuario;
        this.genero = genero;
    }

    public String getNombreUsuario(){
       return this.nombreUsuario;
    }

    public String toString(){
       return this.nombreUsuario + " (" + this.genero + ").";
    }
}
```

Apartado 2 (1,5 puntos)

```
public class Trabajador extends Persona {

      private static int sueldo = 1600;
      private String puesto;

      private final static String MESA = "camarero";
      private final static String COCINA = "cocinero";

      public Trabajador(String puesto, String nombreUsuario, char genero) {
            super(nombreUsuario, genero);
            if (!puesto.equals(MESA) && !puesto.equals(COCINA)) {
                puesto = MESA;
            }
            this.puesto = puesto;
      }

      public void tarea() {
            if (this.puesto.equals(MESA)) {
                System.out.println(super.getNombreUsuario() + ", sirvo las mesas.");
                                    // o this.getNombreUsuario()
            } else {
                System.out.println(super.getNombreUsuario() + ", preparo la
                                    comida."); // o this.getNombreUsuario()
            }
      }
}
```

Apartado 3 (1,5 puntos)

```
public class ClienteVIP extends Persona {
      private Tarjeta tarjeta;
```



```
	public ClienteVIP(String nombreUsuario, char genero,
					int numero, String creacion) {
		super(nombreUsuario, genero);
		this.tarjeta = new Tarjeta(numero, creacion);
	}

	public String toString() {
		return super.toString() + " Tarjeta número " +
				this.tarjeta.getNumero() + " creada el " +
				this.tarjeta.getCreacion();
	}

	public void tarea() {
		System.out.println("Cliente VIP, vengo a comer aquí.");
	}
}
```

Apartado 4 (1 punto)

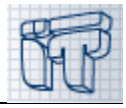
```
public class TestRestaurante {

	public static void main(String[] args) {
		Trabajador t1 = new Trabajador("cocinero", "EvaGarcia_3", 'f');
		ClienteVIP c1 = new ClienteVIP("AndreaIzq", 'f', 12345, "15/03/18");

		ArrayList<Persona> lista = new ArrayList<Persona>();
		lista.add(t1);
		lista.add(c1);

		for (int i=0; i< lista.size(); i++) {
			System.out.println(lista.get(i).toString());
		}


	}

}
```

Apartado 1 (1 punto): Persona

- 0,25: Declaración de clase abstracta implementando GestionRestaurante
  - Si no es abstracta, penalizar -0,1
- 0,25: Atributos y constructor con visibilidad correcta
  - 0,10 Atributos
  - 0,15 Constructor
- 0,25: Método getNombreUsuario().
- 0,25: Método toString( ).
- Los errores significativos están sujetos a sanciones adicionales

Apartado 2 (1,5 puntos): Trabajador

- 0,25: Declaración de clase que extiende la clase Persona
  - Si implementa la interfaz GestionRestaurante, máx 0,15
- 0,25: Declaración de las constantes MESA y COCINA
- 0,65: Atributos y constructor
  - 0,3 Variable estática sueldo y puesto



    - 0,2 Llamada al constructor padre mediante super
    - 0,15 Comprobación de los valores de puesto, y asignación al atributo
    - Si el orden de los parámetros de entrada no se corresponde, penalizar -0,25
- 0,35: Método tarea( )
    - Si no usa getNombreUsuario(), máx 0,15
- Los errores significativos están sujetos a sanciones adicionales

Apartado 3 (1,5 puntos): ClienteVIP

- 0,2: Declaración de clase que extiende la clase Persona
    - Si implementa la interfaz GestionRestaurante, penalizar -0,1
- 0,25: Declaración de atributo tarjeta
- 0,55: Constructor
    - 0,25 Llamada al constructor padre mediante super
    - 0,3 Inicialización del objeto Tarjeta y asignación al atributo correspondiente
- 0,3: Método toString( ) que devuelve un String
    - Si no usa super, máx 0,2
    - Si no usa getters, máx 0,1
- 0,2: Método tarea( )
- Los errores significativos están sujetos a sanciones adicionales

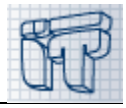
Apartado 4 (1 punto): TestRestaurante

- 0,1: Declaración de clase
- 0,1: Declaración del método principal
- 0,2: Inicialización de los objetos (un cocinero y un cliente) de acuerdo al output
- 0,25: Creación del ArrayList de objetos de tipo Persona
- 0,15: Añadir los objetos al ArrayList
- 0,2: Bucle que recorre el ArrayList y muestra la información por pantalla
- Los errores significativos están sujetos a sanciones adicionales

PROBLEMA 2

Apartado 1 (1 punto)

```
public class Unknown {
	public static int method(String a){
		int value = Integer.parseInt(a);
		int result = 0;
		while(value != 0){
			result += value % 10;
			value = value / 10;
		}
		return result;
	}
}
```



Apartado 2 (1 punto)

```
@RunWith(Parameterized.class)
public class TestUnknown2 {

	@Parameters
	public static Collection<Object[]> data(){
		Object[][] data = new Object[][]{
			{"22",4},{"1234",10},{"11111", 5},{"00000", 0},{"abc", -1}};
		return Arrays.asList(data);
	}

	@Parameter(1)
	public int a;

	@Parameter(0)
	public String b;

	@Test
	public void test(){
		try{
			assertEquals(Unknown.method(b), a);
		}catch(NumberFormatException e){
			assertEquals(a, -1);
		}
	}
}
```

Apartado 1 (1 punto)

- 0.1: Declaración de la clase con su nombre correcto
- 0,2: Declaración del método. Si no es estático, entonces 0.
- 0,7: Funcionamiento correcto del método
  - 0,1 Conversión de String a entero correcta
  - 0,1 Uso adecuado del bucle
  - 0,1 Se usa adecuadamente el return para devolver resultado
  - 0,4 Funcionamiento del método (ej.., hacer las divisiones y los módulos adecuados para la suma)
  - Si hacen throws NumberFormatException, al no ser necesario, penalizar 0.05
  - Si en el código ponen alguna condición para lanzar excepción con throw new NumberFormatException o meten un try/catch no necesario, penalizar 0.2.
- Los errores significativos están sujetos a sanciones adicionales

Apartado 2 (1 punto)

- 0,2: Completar correctamente el objeto data
- 0,2: Completar los tipos de los parámetros
- 0,6: Método test()
  - 0.25 AssertEquals para el caso general
  - 0,20 Manejo de la excepción NumberFormatException para el último caso
  - 0,15 AssertEquals para el caso de la excepción
  - Si los parámetros a y b se ponen al revés, penalizar 0.15
- Los errores significativos están sujetos a sanciones adicionales