NOMBRE:
APELLIDOS:
NIA:
GRUPO:

Primer parcial

2ª Parte: Problemas (7 puntos sobre 10)

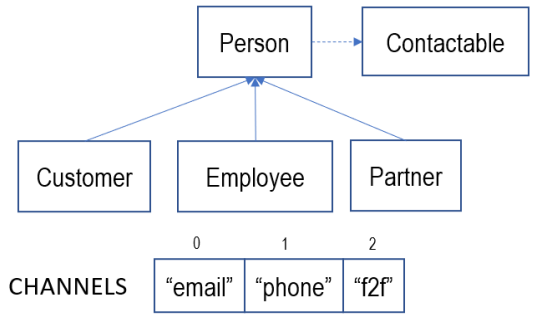
Duración: 70 minutos
Puntuación máxima: 7 puntos
Fecha: 25 de marzo de 2021

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.

Ejercicio 1 (5/ 7 puntos)

El programa de gestión del almacén que has creado para tu proyecto trabaja con 3 tipos de personas: los clientes (Customer), los empleados (Employee) y los colaboradores de otras empresas (Partner). Por tanto se han creado tres tipos diferentes de colaboradores (personas) que suponen una especialización de la clase Person siguiendo la jerarquía que aparece en la figura.

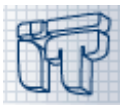



```
public abstract class Person implements Contactable {
    private String firstName;
    private String lastName;
    private int age;
    public Person(String firstName, String lastName, int age) {
        setFirstName(firstName);
        setLastName(lastName);
        setAge(age);
    }
    //setters & getters
}
```

Se pide:

Interfaz Contactable (2/ 5 puntos)

- Implementa la interfaz `Contactable` que permite modelar los contactos de esta y otras aplicaciones.



- Esta interfaz tiene una constante numérica para representar cada tipo de canal: `0-EMAIL, 1-PHONE, 2-F2F` que representa los contactos cara a cara (face to face). Una constante `CHANNELS` de tipo array de Strings con los nombres de cada uno de los canales (ver figura).
- Un método sobrecargado que puede recibir como parámetros
    - En un caso la persona que recibe el mensaje, y el mensaje en formato String
    - En el segundo caso además de los dos parámetros anteriores recibe un entero que representa el canal por el que se envía el mensaje.

Clase Person

- La clase Person tiene dos métodos contactedBy con diferente número de parámetros.
    - Esta clase no tiene información suficiente para implementar el método `contactedBy(Person sender, String msg)`
    - Implementa el método `contactedBy(Person sender, String msg, int channel)` de la clase Person (ver figura) que en la clase en la que se pueda implementar imprime un mensaje con este formato. "Sending msg from: `Pedro` to: `Maria` msg: `Hola` by `email`" suponiendo que el nombre del emisor fuese Pedro, el del receptor María, el mensaje "Hola" y el canal email. No utilices literales para los nombres de los canales. Deberás utilizar los valores del array CHANNELS.
- Implementa el método toString de la clase person que devuelve el nombre, los apellidos y la edad de la persona en el siguiente formato: `Pedro Perez 19`

Clase Employee. Declaración de clase y atributos

Declara la clase `Employee` respetando la jerarquía de la figura teniendo en cuenta las siguientes especificaciones.

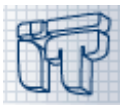
- Empleado (`Employee`) es una especialización de la clase Persona (`Person`)
- Cada empleado tiene un identificador numérico (`employeeID`) que se asigna automáticamente en el momento de su creación. El identificador coincide con el número de empleados existentes contando el recién creado (`numEmployees`)
- El número de empleados existentes (`numEmployees`) es el mismo para todos los objetos de la clase y se encarga de llevar la cuenta de todos los empleados existentes hasta la fecha.

Clase Employee. Constructores y métodos

- Implementa un constructor para la clase `Employee` que reciba los mismos parámetros que su clase padre y los necesarios para asignar valor a todos los atributos de la clase `Employee`
- Implementa el método `contactedBy(Person sender, String msg)`
- Implementa un método toString que devuelve la información del empleado en el siguiente formato `Employee num: 3 Pedro Perez 19` siendo `3` el identificador del empleado y `19` su edad

Clase TestContactables

- Declara la clase `TestContactables` y su método main
- Crea un objeto (`sender`) de tipo empleado que sea el emisor del mensaje.
- Crea un mensaje (`msg`) con el siguiente contenido "`The store will be closed in August`"



- Crea un array (`myContacts`) que contenga tres contactables: un cliente, un empleado y un colaborador.
- Imprime la información de cada uno de los contactos y el resultado de llamar al método `contactedBy` utilizando como emisor el objeto `sender` y como receptores cada uno de los elementos del array `myContacts`. El resultado esperado sería:

```
Customer num: 1 Carolina Casado 28
Sending msg from: Sara to: Carolina msg: The store will be closed in August by: phone
Employee num: 2 Eduardo Estevez 30
Sending msg from: Sara to: Eduardo msg: The store will be closed in August by: email
Partner num: 1 Pablo Perez 25
Sending msg from: Sara to: Pablo msg: The store will be closed in August by:f2f
```

Ejercicio 2 (0,5 / 7 Puntos)

- Dado el siguiente Test para probar la clase `Person` indica el código que falta en los huecos indicados

```
class PersonTest {
    Employee carolina = new Employee("Carolina", "Casado" , 28);
    (1)
    void testPerson() {
        (2)(28, carolina.getAge());
    }
}
```

Ejercicio 3 (1,5 / 7 Puntos)

- Dado el siguiente código, programa el método recursivo
  `public int sum (int [] elements, int pos)`
  que comenzando por el final del array devuelve la suma de los elementos de éste.

```
public class Recursion {

    public static int sum(int[] elements, int pos) {
        //tu codigo aquí
    }

    public static void main(String[] args) {
        int[] elements = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 };

        int sum = sum(elements, elements.length - 1);
        System.out.println(sum);
    }
}
```

SOLUCIÓN DE REFERENCIA (Varias soluciones son posibles)

Ejercicio 1. Solución (5 / 7 puntos)

- **Interfaz contactable (1 punto**)
  - 0,25: Declaración interfaz
  - 0,25: Declaración array de constantes
  - 0,25: Constantes
  - 0,25: Métodos

```
public interface Contactable {
    static final String[] CHANNELS = { "email", "phone", "f2f" };
    static final int EMAIL = 0;
    static final int PHONE = 1;
    static final int F2F = 2; // face to face
    void contactedBy(Person receiver, String msg, int channel);
    void contactedBy(Person receiver, String msg);
}
```

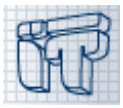
- **Clase Person (1 punto)**
  - contactedBy(Person sender, String msg, int channel)
    - 0,25 declaración
    - 0,25 System.out.println y datos del mensaje salvo CHANNEL
    - 0,25 llamada correcta a Contactable.CHANNELS[channel]
  - toString: 0,25

```
public void contactedBy(Person sender, String msg, int channel) {
    System.out.println("Sending msg from: " + sender.getFirstName() +
                       " to: " + getFirstName() +
                       " msg: " + msg +
                       " by: " + Contactable.CHANNELS[channel]);

}

public String toString() {
    return getFirstName() + " " + getLastName()+ " " + getAge();
}
```

- **Clase Employee (1,75 puntos)**
  - 0,25 Declaración de clase
  - 0,25 Atributos
  - 0,5 Constructor
  - 0,5 contactedBy(Person sender, String msg)
  - 0,25: toString



```
public class Employee extends Person {
    private static int numEmployees;
    private int employeeID;
    public Employee(String firstName, String lastName, int age) {
        super(firstName, lastName, age);
        employeeID = ++numEmployees;
    }
    public void contactedBy(Person sender, String msg) {
        contactedBy(sender, msg, Contactable.EMAIL);
    }
    public String toString() {
        return "Employee num: " + employeeID +
               " " + super.toString();
    }
}
```

- **Clase TestContactables (1,25 puntos)**
  - 0,25: Declaración clase y main
  - 0,25: Creación objetos sender, msg
  - 0,25: Creación array
  - 0,25: Recorrido e imprimir contactos
  - 0,25: Llamada a contactedBy

```
public class TestContactables {
    public static void main(String[] args) {
        Employee sender = new Employee("Sara", "Sainz", 23);
        String msg = "The store will be closed in August";
        Person[] myContacts = { new Customer("Carolina", "Casado", 28),
                                new Employee("Eduardo", "Estevez", 30),
                                new Partner("Pablo", "Perez", 25) };
        for (int i = 0; i < myContacts.length; i++) {
            System.out.println(myContacts[i]);
            myContacts[i].contactedBy(sender, msg);
        }
    }
}
```

Ejercicio 2. Solución (0,5 / 7 puntos)

Rúbrica

- 0.25: @Test
- 0,25: assertEquals

Solución

```
import static org.junit.jupiter.api.Assertions.*;

class PersonTest {
    Employee carolina = new Employee("Carolina", "Casado" , 28);
    @Test
    void testPerson() {
        assertEquals(28, carolina.getAge());
    }
}
```

Ejercicio 3. Solución (1,5 / 7 Puntos)

Rúbrica:

- 0,5 puntos: condición de salida correcta
- 0,5 puntos: llamada recursiva correcta (valores de los parámetros)
- 0,5 puntos: suma correcta.
- 0 si no es recursivo

Solución:

```
public class Recursion {

    public static int sum(int[] elements, int pos) {
        int result = 0;
        if (pos >= 0) {
            result = elements[pos] + sum(elements, pos - 1);
        } else {
            result = 0;
        }
        return result;
    }

    public static void main(String[] args) {
        int[] elements = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 };

        int sum = sum(elements, elements.length - 1);
        System.out.println(sum);
    }
}
```