Universidad Rey Juan Carlos
Escuela Técnica Superior en Ingeniería Informática
19 de enero del 2023

Duración: 2 horas y 15 minutos.
Lea atentamente todo el enunciado antes contestar.
Cualquier suposición o decisión sobre el enunciado del examen debe ser detallada y justificada convenientemente.

Queremos hacer un programa que nos ayude a realizar un sorteo del amigo invisible. Un sorteo (clase Sorteo) tiene un nombre, la fecha actual, un conjunto de participantes, unas restricciones (exclusiones) entre los participantes y el resultado del sorteo.

En el sorteo del amigo invisible participan varias personas (clase Persona) identificadas por su nombre, teléfono y email. El nombre y el email serán campos obligatorios y en el caso del email solo se admitirán cuentas de Gmail (dominio gmail.com). En caso de que alguno de los dos campos obligatorios no sea correcto se deberá crear y lanzar una excepción propia (clase PersonaIncorrecta) siendo capturada en el sitio correspondiente para ignorar ese participante y seguir con la ejecución del programa.

En el sorteo se pueden añadir restricciones de dos tipos: entre dos personas (clase Pareja), o entre un grupo de personas (clase Familiares). Por ejemplo, si Mario es el novio de María, se podrá definir una exclusión donde se indique que estas dos personas no pueden regalarse entre sí. Es decir, nunca podría salir que Mario tiene que regalar a María o que María tiene que regalar a Mario. Por tanto, se podrá definir una exclusión de tipo pareja definida por las dos personas que lo forman siempre y cuando sean distintas. Se considerará que una persona es igual a otra cuando coincida el correo electrónico sin tener en cuenta si está escrito en mayúsculas o minúsculas.

El otro caso de restricción sería entre un grupo de personas (Familiares). Imaginad que se realiza un sorteo entre un grupo de niños donde hay varios hermanos: Susana, Mikel, y Noah. En este caso, se podría añadir una exclusión donde ninguno de los hermanos pudiera regalar a otro. Por tanto, la clase Familiares tendría un atributo de tipo listado que contuviera aquellas personas que tienen lazos familiares y que, por tanto, no pueden regalarse entre ellos.

Cuando se crea un nuevo sorteo, se debe proveer el nombre del mismo e inicializar la fecha a la actual. Como se ha comentado, se deberán poder añadir participantes al sorteo. Como mínimo debe existir 4 participantes para que se pueda realizar el sorteo. Una vez añadido los participantes, bien se pueden añadir restricciones de parejas o de grupos familiares, o bien podemos realizar el sorteo directamente.

Cuando se terminen de añadir participantes y restricciones (opcional esto último), se realizará el sorteo y se generará un fichero de texto con el nombre dd-MM-yyyy-hh-mm-ss.txt siendo el nombre la fecha del sorteo. Cada fila tendrá el nombre de la persona en primer lugar que regala a otra persona. Los datos deben estar ordenados alfabéticamente por el nombre de la primera persona que regala a otra.

Esta implementación tiene que compilar con el programa principal AmigoInvisible.java. Tal y como se puede observar en este programa, se pueden añadir participantes al sorteo de forma individual o bien

usando el fichero de pruebas datos.txt donde se han definido un conjunto de participantes. En cada una línea de este fichero, aparece el nombre del participante, el teléfono y el email separados por un tabulador.

El diagrama UML que modela las clases de este sistema se puede ver en la siguiente figura.

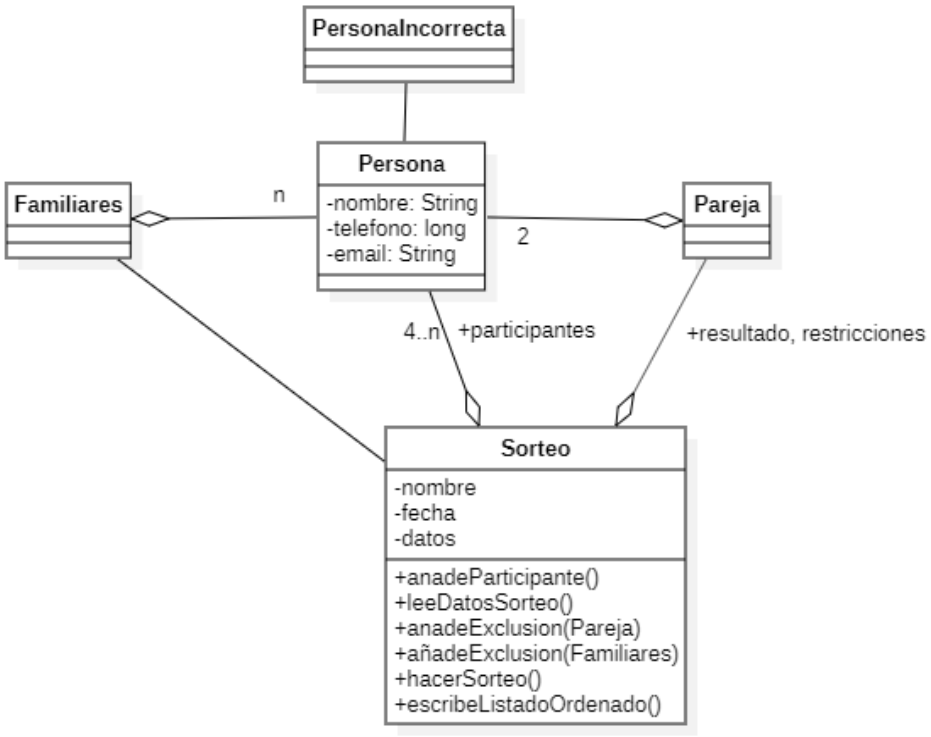

Ilustración 1. UML del examen Amigo Invisible

Se pide:

- Desarrollar al completo la clase Persona. **[2 puntos]**
- Implementar la excepción propia PersonaIncorrecta. **[0,5 puntos]**
- Implementar la clase Pareja. **[1,5 puntos]**
- Implementar la clase Sorteo con los atributos, un constructor que reciba el nombre del sorteo y los métodos siguientes: **[6 puntos]**
    - `anadeParticipante(Persona p)`, añade un participante al sorteo. No se admiten participantes duplicados. **[0,5 puntos]**
    - `leeDatosSorteo(String f)`, recibe el nombre del fichero que hay que leer y procesa los datos de los participantes de cada una de las filas. **[1,25 puntos]**
    - `anadeExclusion(Pareja p)`, realiza una exclusión de una pareja de participantes. **[0,75 puntos]**
    - `anadeExclusion(Familiares f)`, realiza una exclusión sobre un grupo de personas. **[0,75 puntos]**
    - `hacerSorteo()`, método que realiza propiamente el sorteo una vez definidos los participantes, y de forma opcional las restricciones. Dado que la implementación de este método es compleja, se pide realizar este algoritmo, aunque no sea perfecto: realizar el sorteo de forma aleatoria sin tener en cuenta las restricciones, comprobar si el sorteo

realizado es válido (es decir, ninguna persona se regala así misma y ninguna persona regala a alguien que esté en restricciones). Si el sorteo es válido se escribirá el resultado en fichero y si no se devolverá un código de error. **[2 puntos]**

- `escribeListadoOrdenado ()`, método que genera el fichero de texto con el resultado del sorteo. **[0,75 puntos]**

Nota: No se deben implementar los métodos get() / set() ni los constructores que no se piden explícitamente en el enunciado.

Nota 2: Se facilita:

- El programa principal con el que tiene que compilar el código implementado
- La clase Familiares que debe incluirse en el proyecto para su correcto funcionamiento
- El fichero de entrada de datos que debe usarse para probar la lectura de un fichero de texto

Nota 3: Se facilita la siguiente información sobre clases de utilidad que se necesitan para la implementación requerida en este examen:

- int aleatorio = (int)(Math.random() * num); → Aleatorio contendrá un número entero generado al azar entre 0 y num-1 (ambos incluidos).
- Instrucción: DateTimeFormatter.ofPattern("dd-MM-yy") que permite formatear una fecha de acuerdo a un determinado patrón.
- Clase LocalDateTime, clase a usar para la generación de fechas e instantes de tiempo.
- Clase Scanner, usada para la lectura de ficheros línea a línea.
- Clase File, clase que gestiona ficheros en disco duro.
- Clases BufferedWriter y FileWriter que permiten realizar la escritura de ficheros de texto.

Clase AmigoInvisible [0 puntos – facilitada en el enunciado]

```
public class AmigoInvisible {
    public static final String FENTRADA = "datos.txt";
    public static void main(String[] args) {
        Sorteo sorteo = new Sorteo("Mi primer sorteo");
        sorteo.leeDatosSorteo(FENTRADA);

        Persona p1 = new Persona("P1",123456789, "p1@gmail.com");
        Persona p2 = new Persona("P2",23456789, "p2@gmail.com");
        Persona p3 = new Persona("P3",3456789, "p3@gmail.com");
        Persona p5 = new Persona("P5",56789, "p5@gmail.com");
        Persona p6 = new Persona("P6",6789, "p6@gmail.com");
        Persona p7 = new Persona("P7",789, "p7@gmail.com");
        Persona p8 = new Persona("P8",89, "p8@gmail.com");

        sorteo.anadeParticipante(p1);
        sorteo.anadeParticipante(p2);
        sorteo.anadeParticipante(p3);
```

```
        sorteo.anadeParticipante(p5);
        sorteo.anadeParticipante(p6);
        sorteo.anadeParticipante(p7);
        sorteo.anadeParticipante(p8);

        Pareja par1 = new Pareja (p1, p3);
        Familiares f = new Familiares();
        f.anadeFamiliar(p2);
        f.anadeFamiliar(p7);
        f.anadeFamiliar(p8);

        sorteo.anadeExclusion(par1);
        sorteo.anadeExclusion(f);

        if (sorteo.hacerSorteo())
           System.out.println("El sorteo se ha realizado");
        else
           System.out.println("Hubo un problema en el sorteo. Quite restricciones");

    }
}
```

Clase Familiares [0 puntos – facilitada en el enunciado]

```
public class Familiares {
    private ArrayList<Persona> lista;

    public Familiares(){
        lista = new ArrayList<>();
    }

    public ArrayList<Persona> getListado(){ return lista;}

    public boolean anadeFamiliar(Persona p){
        if (!lista.contains(p))
            return lista.add(p);
        return false;
    }
}
```

Solución

Clase Persona [2 puntos]

```
public class Persona implements Comparable{
    private String nombre;
    private long telefono;
    private String email;
    private static final String DOMINIO = "@gmail.com";

    public Persona(String n, long t, String e){
        this(n,e);
        telefono = t;
    }

    public Persona(String n, String e) throws PersonaIncorrecta{
        if ((n == null) || (n.isBlank()) || emailIncorrecto(e))
            throw new PersonaIncorrecta("Nombre o email inválidos");
        nombre = n;
        email = e;
    }

    public String getNombre() { return nombre; }
    public long getTelefono() { return telefono; }
    public String getEmail() { return email; }

    public void setNombre(String n){
        if ((n == null) || (n.isBlank())) throw new PersonaIncorrecta("Nombre nulo o vacío");
        nombre = n;
    }

    public void setTelefono (long t){ telefono = t; }

    public void setEmail(String e){
        if (emailIncorrecto(e)) throw new PersonaIncorrecta("Email incorrecto");
        email = e;
    }

    private boolean emailIncorrecto(String e){
        return ((e == null) || (e.isBlank()) || (!e.endsWith(DOMINIO)));
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null) return false;
        if (getClass() != o.getClass()) return false;
```

```
        Persona aux = (Persona) o;
        return (aux.getEmail().equalsIgnoreCase(this.getEmail()));
    }

    @Override
    public String toString(){ return nombre;}

    @Override
    public int compareTo(Object p) {
        if (p instanceof Persona)
            return this.nombre.compareTo(((Persona)p).getNombre());
        return 0;
    }
}
```

Clase Pareja [1,5 puntos]

```
public class Pareja implements Comparable{
    private Persona p1;
    private Persona p2;

    public Pareja(Persona p1, Persona p2){
        this.p1 = p1;
        this.p2 = p2;
    }

    public Persona getP1(){ return p1; }
    public Persona getP2(){ return p2; }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null) return false;
        if (getClass() != o.getClass()) return false;
        Pareja aux = (Pareja) o;
        return ((aux.getP1().equals(this.getP1())) && (aux.getP2().equals(this.getP2())));
        // Quitada la restricción recíproca
    }

    @Override
    public int compareTo(Object p){
        if (p instanceof Pareja)
            return p1.compareTo(((Pareja) p).getP1());
        return 0;
    }
}
```

Clase PersonaIncorrecta [0,5 puntos]

```
public class PersonaIncorrecta extends RuntimeException{
    public PersonaIncorrecta () { super();}
    public PersonaIncorrecta (String mensaje){ super(mensaje); }
}
```

Clase Sorteo [6 puntos]

```
public class Sorteo {
    private final String nombre;
    private final LocalDateTime fecha;
    private ArrayList<Persona> participantes;
    private ArrayList<Pareja> resultado;
    private ArrayList<Pareja> restricciones;
    private static final int MINPARTICIPANTES = 4;

    public Sorteo(String n) {
        nombre = n;
        fecha = LocalDateTime.now();
        participantes = new ArrayList<>();
        resultado = new ArrayList<>();
        restricciones = new ArrayList<>();
    }

    public void anadeParticipante(Persona p) {
        if (!participantes.contains(p))
            participantes.add(p);
    }

    public boolean leeDatosSorteo(String f){
        try{
            int linea = 1;
            Scanner ent= new Scanner(new File(f));
            while (ent.hasNextLine()){
                String[] campos = ent.nextLine().split("\t");
                try{
                    Persona p = new Persona(campos[0], Integer.valueOf(campos[1]), campos[2]);
                    anadeParticipante(p);
                    linea++;
                }
                catch(PersonaIncorrecta pi){
                    System.err.println("Error en la linea " + linea + " del fichero de datos");
                }
            }
            ent.close();
        }
```

```
        catch (IOException ex){
            System.err.println("Error al leer los datos del sorteo" + ex);
            return false;
        }
        return true;
    }

    public boolean anadeExclusion(Pareja p){
        Pareja inversa = new Pareja(p.getP2(), p.getP1());
        boolean res = true;

        if (!restricciones.contains(p)){
            res = restricciones.add(p);
        }

        if ((res) && (!restricciones.contains(inversa))){
            return restricciones.add(inversa);
        }

        return false;
    }

    public boolean anadeExclusion(Familiares f){
        ArrayList<Persona> ltemp = new ArrayList(f.getListado());
        boolean error = false;

        for (int i = 0; !error && i < ltemp.size() ; i++){
            for (int j = i + 1; !error && j < ltemp.size() ; j++){
                Pareja aux = new Pareja(ltemp.get(i), ltemp.get(j));
                if (!anadeExclusion(aux)) error = true;
            }
        }

        return error;
    }

    public boolean hacerSorteo(){
        if (participantes.size() < MINPARTICIPANTES) return false;

        boolean valido = true;
        ArrayList<Persona> temp = new ArrayList<>(participantes);
        resultado = new ArrayList<>();
        for (Persona p : participantes) {
                int randomIndex = (int)(Math.random() * temp.size());
                Persona res = temp.get(randomIndex);
                Pareja aux = new Pareja(p, res);
```

```
                resultado.add(aux);
                temp.remove(randomIndex);
        }

        Pareja p;
        Iterator <Pareja> it = resultado.iterator();
        while (it.hasNext()&& valido){
          p = it.next();
          if ((p.getP1().equals(p.getP2())) || (estaEnRestricciones(p)))
              valido = false;
        }

        if (valido) escribeListadoOrdenado();
        return valido;
    }

    private boolean estaEnRestricciones(Pareja p){
        return restricciones.contains(p);
    }

    public void escribeListadoOrdenado (){
        String f = fecha.format(DateTimeFormatter.ofPattern("dd-MM-yyyy-hh-mm-ss"));

        File fichero = new File(f + ".txt");
        if (!fichero.exists()){
            try {
                BufferedWriter bw = new BufferedWriter(new FileWriter(f + ".txt"));
                bw.write(this.toString());
                bw.close();
            } catch (IOException ex) {
                System.err.println("Error al generar el listado ordenado " + ex);
            }
        }
    }

    @Override
    public String toString(){
        String txt = "";
        Collections.sort(resultado);
        for (Pareja par: resultado){
            txt += par.getP1() + "-->" + par.getP2() + "\n";
        }
        return txt;
    }
}
```