Academia Java - Noviembre 2022

Correo *

gastonrider16@gmail.com

Nombre y Apellido *

Gastón Ramirez

Ejercicios

Resuelva los siguientes ejercicios en los campos de texto provistos.
Recuerde que no es necesario resolver todos los ejercicios.

Consideraciones

1. Si bien el enunciado hace referencia a Java, se puede resolver con cualquier otro lenguaje de programación orientado a objetos ACLARANDO QUÉ LENGUAJE FUE USADO. En el caso de lenguajes multiparadigma, utilizar solo los mecanismos orientados a objetos.
2. Se recomienda que los ejercicios sean resueltos en un editor local para evitar perdida de datos por fallas de Internet. No es necesario que sea una IDE, se puede resolver en un editor de texto simple.
3. Es importante la legibilidad del código, será leído por un humano.
4. Solo implemente lo pedido en los enunciados. Asuma que el resto del código existe.

IMPORTANTE: Utilice el lenguaje orientado a objetos con el que se sienta más cómodo. El objetivo de la evaluación no es evaluar los conocimientos de un lenguaje en particular, sino el conocimiento del paradigma de programación orientada a objetos.

¿Qué lenguaje de programación utilizarás para resolver los ejercicios?

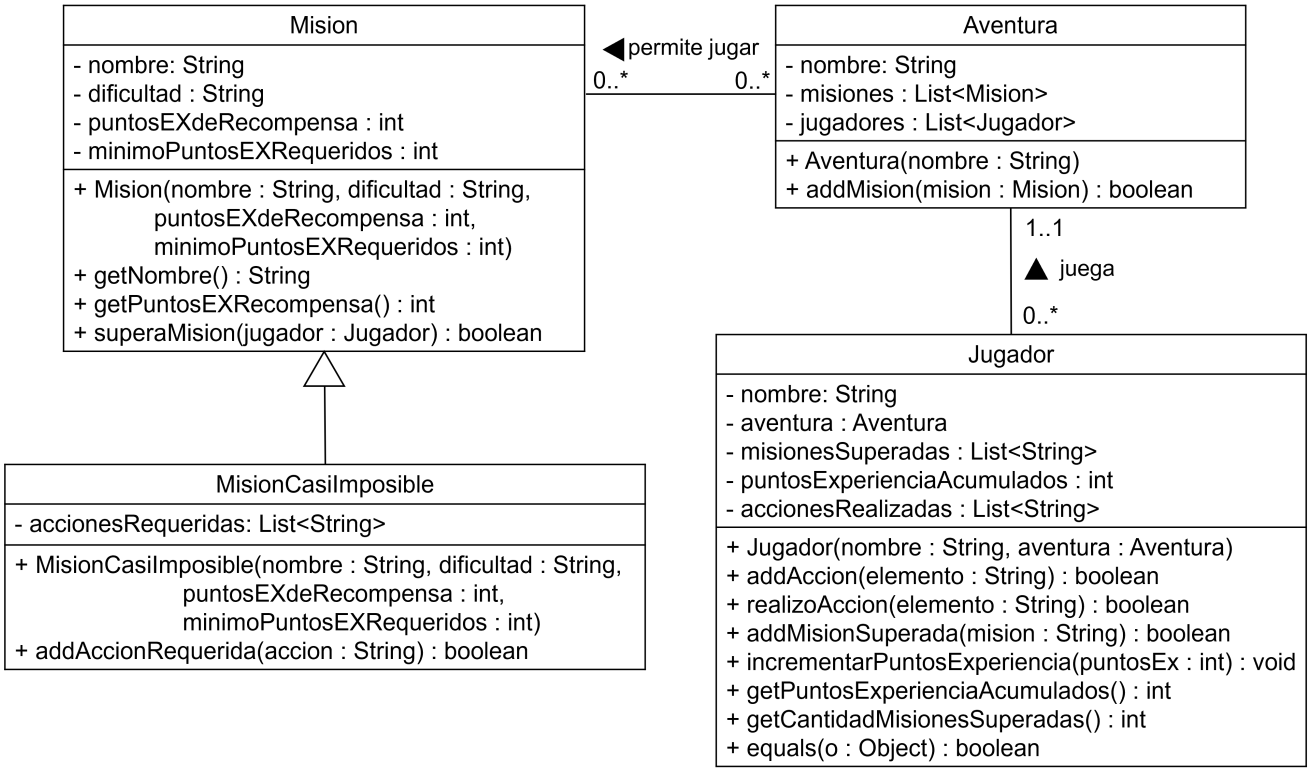
- ○ C#
- ○ C++
- ◉ Java
- ○ JavaScript
- ○

- Kotlin
- Python
- Otro: .....

Narrativa

Se tiene un sistema que modela un juego. Hasta el momento, el sistema administra jugadores y misiones. Cada jugador conoce su nivel de experiencia, acciones que realizó y misiones que superó. Por su lado, las misiones registran el nombre, dificultad, puntos de experiencia de recompensa y puntos de experiencia requeridos para realizar la misión. Además las misión es responsable de saber si un jugador supera la misión o no, basado solamente en los puntos de experiencia.
El administrador de juego desea agregar otro tipo de misiones, llamadas misiones casi imposibles, donde además de los puntos de experiencia, se desea realizar otros controles adicionales sobre el jugador. El diseñador modificó el diagrama y nos dejó instrucciones para implementar esta nueva característica del sistema.

Diagrama



NOTA

Si bien el enunciado hace referencia a Java, se puede resolver con cualquier otro lenguaje de programación orientado a objetos ACLARANDO QUÉ LENGUAJE FUE USADO. En el caso de lenguajes multiparadigma, utilizar solo los mecanismos orientados a objetos.

Ejercicio 1

Tenemos un diagrama de clases preliminar de un sistema y hay que implementar la clase **MisionCasiImplosible** de acuerdo con el diagrama. El resto de las clases serán implementadas por otros desarrolladores.

addAccionRequerida(elemento: String): boolean solo agrega una acción requerida si esta no se encuentra en la lista de acciones requeridas. Tener en cuenta que el método debe retornar **true** si la acción pudo ser agregada y **false** en caso contrario.

Una vez que se haya implementado la clase, se ejecutará el siguiente código, el cual no debe tener errores de ejecución:

```
MisionCasiImposible m1 = new MisionCasiImposible("Mision CI", "difícil", 100, 200);
m1.addAccionRequerida("llegar al castillo");
```

Implemente la solución al Ejercicio 1 en el siguiente espacio

```
public class MisionCasiImposible extends Mision {
   public List<String> getAccionesRequeridas() {
      return accionesRequeridas;
   }

   private List<String> accionesRequeridas = new ArrayList<>();
   public MisionCasiImposible(String nombre, String dificultad, int
puntosEXdeRecompensa, int minimoPuntosEXRequeridos){
      this.setNombre(nombre);
      this.setDificultad(dificultad);
      this.setPuntosEXdeRecompensa(puntosEXdeRecompensa);
      this.setMinimoPuntosEXRequeridos(minimoPuntosEXRequeridos);
   }
   public boolean addAccionRequerida(String elemento){
      if(!this.existeAccionRequerida(elemento)){
         accionesRequeridas.add(elemento);
         return true;
      }
      else{
         return false;
      }
   }

   boolean existeAccionRequerida(String elemento){
      return accionesRequeridas.contains(elemento);
   }

   public static void main(String[]args){
      MisionCasiImposible m1 = new MisionCasiImposible("Mision CI", "dificil", 100,
200);
      m1.addAccionRequerida("llegar al castillo");
   }
}
```

En este espacio podés dejar comentarios sobre el Ejercicio 1.

En el constructor, MisionCasiImposible usa los setters de Mision puesto que los atributos de Mision son privados y los puede usar porque MisionCasiImposible hereda de Mision. Asigna los parametros del contructor a los atributos. Para public boolean addAccionRequerida(String elemento), en el if delegue la responsabilidad de saber si existe una accion requerida. Si no existió una accion requerida en la lista, ejecuta accionesRequeridas.add(elemento); y retorna true, de lo contrario retorna false

Ejercicio 2

En principio, para superar una **Mision**, el Jugador tiene que haber acumulado más puntos de experiencia que el **minimoPuntosEXRequeridos** requerido por la misión. Para ello, definieron el siguiente método en la clase **Mision**:

```
public boolean superaMision(Jugador jugador){
	return minimoPuntosEXRequeridos < jugador.getPuntosExperienciaAcumulados();
}
```

Ahora, para superar una **MisionCasiImposible** se requiere cumplir más condiciones. En este caso, el **Jugador** no solo debe haber acumulado más puntos de experiencia que el mínimo requerido por la misión, sino que también debe haber realizado todas las acciones requeridas por la misión (representadas por **accionesRequeridas: List<String>**). En caso de que la segunda condición no se cumpla, para superar la **MisionCasiImposible** la cantidad de acciones que le faltan cumplir al jugador debe ser menor a la cantidad de misiones superadas por el **Jugador.**

Tenga en cuenta el siguiente método de la clase **Jugador**:

```
public boolean realizoAccion(String elemento){
	return accionesRealizadas.contains(elemento);
}
```

Implemente los métodos necesarios para incorporar la modificación solicitada. Indique a qué clases pertenecen los métodos implementados.

Implemente los métodos necesarios para resolver el Ejercicio 2. Indique a qué clases pertenecen los métodos implementados.

```
public class Mision {
   private String nombre;
   private String dificultad;
   private int puntosEXdeRecompensa;
   private int minimoPuntosEXRequeridos;

   public Mision(String nombre, String dificultad, int puntosEXdeRecompensa, int
minimoPuntosEXRequeridos){
      this.nombre = nombre;
```

```
        this.dificultad = dificultad;
        this.puntosEXdeRecompensa = puntosEXdeRecompensa;
        this.minimoPuntosEXRequeridos = minimoPuntosEXRequeridos;
    }

    public boolean superaMision(Jugador jugador){
        return minimoPuntosEXRequeridos < jugador.getPuntosExperienciaAcumulados
();
    }

    public boolean superaMisionCasiImposible(Jugador jugador, MisionCasiImposible
misionCasiImposible){
        if(this.superaMision(jugador)){
            if(misionCasiImposible.jugadorCumplioTodasAccionesRequeridas(jugador)){
                return true;
            }
            else {
                return misionCasiImposible.getAccionesRequeridas().size() <
jugador.getAccionesRealizadas().size();
            }
        }
        return false;
    }
//..
}

public class Jugador {
    private String nombre;
    private Aventura aventura;
    private List<String> misionesSuperadas;
    private int puntosExperienciaAcumulados;

    private List<String> accionesRealizadas;

    public Jugador(String nombre, Aventura aventura){
        this.nombre = nombre;
        this.aventura = aventura;
    }

    public boolean realizoAccion(String elemento){
        return accionesRealizadas.contains(elemento);
    }
//..
}
```

En este espacio podés dejar comentarios sobre el Ejercicio 2.

En Mision, implemente el método superaMisionCasiImposible(Jugador jugador, MisionCasiImposible misionCasiImposible) que su primera condición es la misma que el método superaMision(Jugador jugador), pero en caso que se cumpla hay otra condicion que si el jugador cumplió todas las misiones, devolvera true, caso contrario, devolverá misionCasiImposible.getAccionesRequeridas().size() < jugador.getAccionesRealizadas().size();

Ejercicio 3a

Una vez implementados las **Misiones** y **MisionesCasiImposibles**, se quiere implementar la funcionalidad correspondiente a agregar un **Jugador** a una **Aventura**. Para ello quieren implementar el método **public boolean agregarJugador(jugador: Jugador): boolean**. Para poder agregar un **Jugador** a la **Aventura**, este no debe haber sido previamente agregado.

Implementar esa funcionalidad teniendo en cuenta:

```
Si el jugador ya existía:
        Retornar false

Si el jugador pudo ser agregado:
        Por cada misión de la aventura:
            Si el jugador superaMision:
                Agregarle al jugador el nombre de la misión superada.
                Si la misión pudo ser agregada (addMisionSuperada
                retornó true), incrementar los puntos de experiencia del
                jugador de acuerdo a los puntos retornados por la misión.
        Retornar true
```

Implemente los métodos necesarios para resolver el Ejercicio 3a. Indique a qué clases pertenecen los métodos implementados.

```
public class Aventura {
   private String nombre;
   private List<Mision> misiones;
   private List<Jugador> jugadores;
   public Aventura(String nombre){
      this.nombre = nombre;
   }
   public boolean agregarJugador(Jugador jugador){
      if(!this.existeJugador(jugador)){
         misiones.stream().forEach(mision -> {
            if(mision.superaMision(jugador)){
               jugador.setMisionesSuperadas(mision.getNombre());
            }
            if(jugador.addMisionSuperada(mision.getNombre())){
               jugador.setPuntosExperienciaAcumulados(mision.
getPuntosEXdeRecompensa());
            }
         });
         return true;
      }
      else {
         return false;
      }
   }
```

```
    public boolean existeJugador(Jugador jugador){
        return jugadores.contains(jugador);
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
public void setMisiones(Mision mision){
        this.misiones.add(mision);
    }
}

public class Jugador {
    private String nombre;
    private Aventura aventura;
    private List<String> misionesSuperadas;
    private int puntosExperienciaAcumulados;
    private List<String> accionesRealizadas;

   public void setMisionesSuperadas(String mision) {
        this.misionesSuperadas.add(mision);
    }

    public boolean addMisionSuperada(String mision){
        return misionesSuperadas.contains(mision);
    }

    public int getPuntosExperienciaAcumulados() {
        return puntosExperienciaAcumulados;
    }

    public void setPuntosExperienciaAcumulados(int puntosExperienciaAcumulados) {
        this.puntosExperienciaAcumulados = puntosExperienciaAcumulados;
    }

    public List<String> getAccionesRealizadas() {
        return accionesRealizadas;
    }

    public List<String> getMisionesSuperadas() {
        return misionesSuperadas;
    }
}
```

Ejercicio 3b

Describa que fue necesario tener en cuenta para que el código desarrollado en el punto 3a pueda soportar tanto misiones y misiones casi imposible. ¿Cómo se determina que lógica utilizar para saber si se superó la misión?

Respuesta ejercicio 3b

MisionCasiImposible al heredar de Mision, en la clase Aventura su atributo private List<Mision> misiones; sirve para ambas clases. Aventura tiene el método public void setMisiones(Mision mision) que permitira agregar tanto MisionCasiImposible como Mision. Para determinar si se superó una misión, Jugador implemente: public boolean addMisionSuperada(String mision){
return misionesSuperadas.contains(mision);
}

En este espacio podés dejar comentarios sobre el Ejercicio 3.

Para este ejercicio, en la clase Aventura, implemente public boolean agregarJugador(Jugador jugador) que donde si no existe un jugador, me retornará false. De lo contrario, implemente un forEach de la lista misiones, el cual posee 2 ifs que dependiendo de la condición, ejecutará ciertas acciones. El primer if que es if(mision.superaMision(jugador)) si se comple ejecutará jugador.setMisionesSuperadas(mision.getNombre());
En el segundo if que es if(jugador.addMisionSuperada(mision.getNombre())) si se cumple ejecutará jugador.setPuntosExperienciaAcumulados(mision.getPuntosEXdeRecompensa());

Ejercicio 4

En el sistema encontramos el siguiente método en el que no fueron muy claros con los nombres de métodos y variables.

```
public List<String> metodoSinNombre(){
      List<String> variableSinNombre = new ArrayList<>();
      for(Mision m : misiones){
            boolean booleanSinNombre = false;
            for(Jugador j : jugadores)
                  if(m.superaMision(j))
                        booleanSinNombre = true;
            if(!booleanSinNombre)
                  variableSinNombre.add(m.getNombre());
      }
      return variableSinNombre;
}
```

```
public static void main(String[] args) {

	Mision m1 = new Mision("buscando la olla del duende","facil",100,0);
	Mision m2 = new Mision("lobo está?","intermedio",200,100);

	Mision m5 = new MisionCasiImposible("san jorge un poroto","dificil",500,300);
	((MisionCasiImposible)m5).addAccionRequerida("derrotar 10 dragones")

	Aventura j1 = new Aventura("D&D");
	j1.addMision(m1);
	j1.addMision(m5);
	j1.addMision(m2);

	Jugador player1 = new Jugador("Juan",j1);

	Jugador player2 = new Jugador("Sebastián",j1);
	player2.addAccion("derrotar 10 dragones");

	j1.addJugador(player1);
	j1.addJugador(player2);

	System.out.println(j1.metodoSinNombre());

}
```

¿Cuál es la salida de ejecutar dicho código? NOTA: Java imprime las listas de string con el siguiente formato: [String1, String2, String3]

Devuelve una lista de nombres de Mision

Documente la funcionalidad del metodoSinNombre

Primero define el atributo List<String> variableSinNombre.
Segundo realiza un for de las misiones, en donde en su primera linea setea boleanSinNombre como falso.
Dentro del mismo for, despues de su primera linea, surge otro for pero de jugadores.
Dentro del for de jugadores, hay un condicional que si el jugador superó la misión, setea a boleanSinNombre como verdadero.
Afuera del for de jugadores, pero dentro del for de misiones, hay un condicional donde si booleanSinNombre es falso, agregara el nombre de la mision a variableSinNombre.
Finalmente por fuera del for de misiones, retorna v

En este espacio podés dejar comentarios sobre el Ejercicio 4.

Ejercicio 5: Un poco de algoritmia

Dada una lista de enteros y un entero objetivo se debe retornar la lista de todos pares, sin repetir, de índices tal que la suma de los enteros en la lista con esos índices sea igual al valor objetivo.

Por ejemplo, dado:

```
L = [2, 6, 8 , 3, 5, 0, 2, 6]
T = 8
```

el método debe devolver:

```
[(0, 1), (0, 7), (1, 6), (2, 5), (3, 4), (6, 7)]
```

El primer elemento `(0,1)` es porque `L[0] + L[1] = 2 + 6 = 8`.
El segundo elemento `(0,7)` es porque `L[0] + L[7] = 2 + 6 = 8`.
El tercer elemento `(1,6)` es porque `L[1] + L[6] = 6 + 2 = 8`.
El cuarto elemento `(2,5)` es porque `L[2] + L[5] = 8 + 0 = 8`.
…

Nótese que el par `(1,0)` también suma 8, pero cómo se encuentra el par `(0,1)` se considera repetido.

Implemente el código necesario para resolver el Ejercicio 5.

```
public class Calculo {
   public static void main(String[] args){
      int numeros[] = {2,6,8,3,5,0,2,6};
      for(int x=0; x<numeros.length; x++){
         for(int y=1; y<numeros.length; y++){
            if(numeros[x]%2==0 && numeros[y]%2==0){
               if (numeros[x] + numeros[y] == 8){
                  System.out.println(numeros[x]);
                  System.out.println(numeros[y]);
               }
            }
         }
      }
   }
}
```

En este espacio podés dejar comentarios sobre el Ejercicio 5.

En este espacio podés dejar comentarios sobre la evaluación