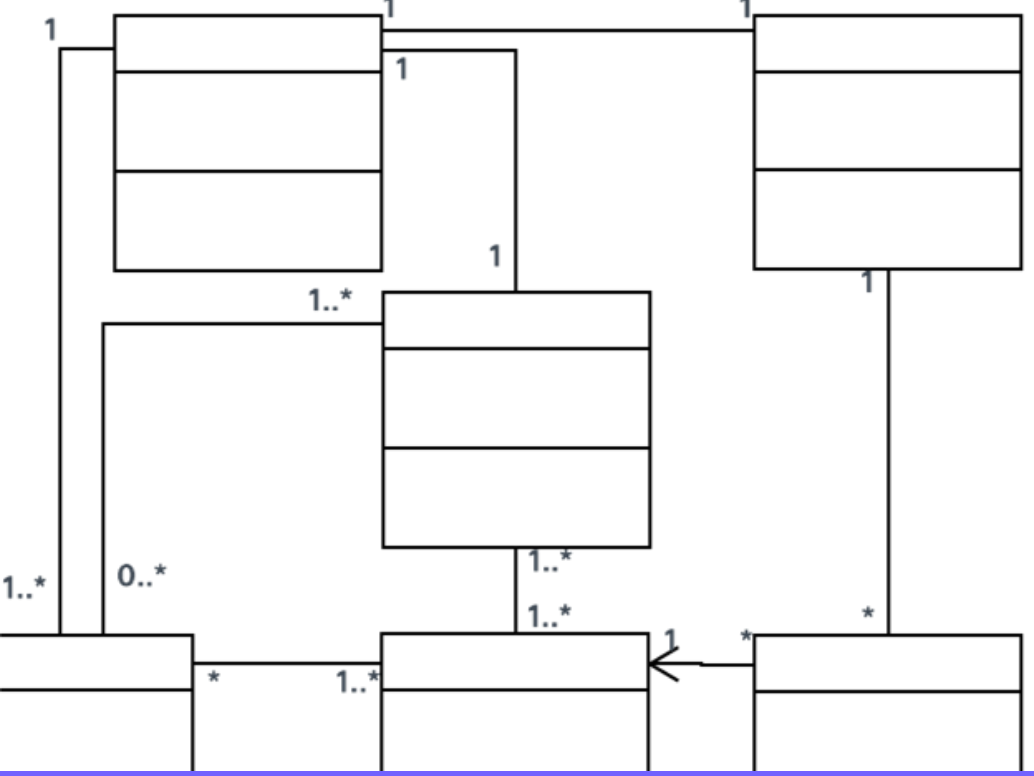

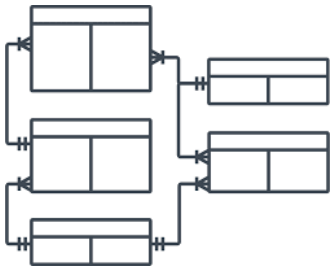

Diseño de Sistemas

GameTrack – Gestión de Videojuegos

Lectura…

Link al enunciado

Requerimiento 1

Permitir gestionar los establecimientos, por parte de un administrador general de la plataforma.

Abstracciones necesarias:

- Clase "***Establecimiento"*** con, mínimamente, un nombre y una colección de máquinas (siguiente requerimiento)

Requerimiento 2

Permitir que el encargado de cada establecimiento pueda gestionar los juegos y las máquinas que se encuentren en él.

Abstracciones necesarias:

- Clase ***"Máquina"***

- Clase ***"Juego"***
  - No es necesaria una herencia porque no existe distinto tipo de comportamiento entre los juegos.
  - El Tipo de Juego puede ser o un enumerado o una clase concreta ("videojuego", "simulador", "destreza", etc. Serían instancias de esta clase).
  - Tampoco es necesaria la herencia en tipos de juego ya que no existe distinto comportamiento.

Requerimiento 3

Permitir la gestión de jugadores y de sus membresías, por parte de los empleados de las sucursales.

Abstracciones necesarias:

- Clase ***"Jugador"***
  - Debería tener como atributo una colección de membresías, para guardar el historial de todas las que tuvo.
  - Debería tener los ***puntos*** acumulados.

- Clase ***"Membresía"***
  - Debería tener un atributo "activo" para saber si es la membresía activa o no.
  - Debería tener los ***créditos*** actuales en esa membresía.

Requerimiento 4

Permitir la gestión de tipos de membresías de acuerdo a las necesidades de mercado.

Se pueden modelar de varias formas, principalmente:

- Clase abstracta "***Membresía***", con Básica, Gold y Premium como clases hijas ***[- flexible]***
  - Deberían tener todos los valores configurables como atributos de clase (statics), ya que se deben poder modificar.

- Clase concreta ***"Membresía"*** que tenga como atributo una clase "***TipoMembresía***" ***[+ flexible]***
  - TipoMembresía debería tener como atributos:
    - Nombre
    - ***Beneficios*** de recarga de crédito
      - Interface "Beneficio", con un método "number aplicarSobre(créditos)", cuyas clases que la implementan podrían ser "BeneficioPorcentual", "BeneficioMultiplicacion", por ejemplo.
    - ***Descuentos*** para consumo de créditos
      - Interface "Descuento", con un método "number aplicarSobre(créditos)", cuyas clases que la implementan podrían ser "DescuentoPorcentual", etc.

Requerimiento 5

Permitir la recarga de crédito para la membresía de los jugadores, por parte de los empleados de las sucursales.

Abstracciones necesarias:

- Método "***recargarCrédito***" en el jugador y membresía.
- Se debería tener en cuenta el tipo de membresía para saber si aplican, o no, beneficios.

Recargar una cierta cantidad de créditos

```
>> Jugador
        recargarCredito(cant) {
                this.membresiaActual().recargarCredito(cant);
        }
```

ALTERNATIVA MEMBRESÍAS FLEXIBLES

```
>> Membresia
        recargarCredito(cant) {
                this.creditos += this.tipo.beneficios.map(b -> b.aplicarSobre(cant)).sum();
        }

>> BeneficioMultiplicacion implements Beneficio
        aplicarSobre(cant) {
                return cant * this.multiplicador
        }

>> BeneficioMultiplicacionTopeMax implements Beneficio
        aplicarSobre(cant) {
                retrun min(cant * this.multiplicador, this.topeMax)
        }

>> BeneficioPorcentual
        aplicarSobre(cant) {
                return cant * (this.porcentaje/100);
        }
```

Recargar una cierta cantidad de créditos

```
>> Jugador
        recargarCredito(cant) {
                this.membresiaActual().recargarCredito(cant);
        }
```

OTRA ALTERNATIVA DE MEMBRESÍAS

```
>> abstract Membresia
        abstract recargarCredito(cant);

>> Basica extends Membresia
        recargarCredito(cant) {
                this.credito += cant;
        }

>> Gold extends Membresia
        recargarCredito(cant) {
                this.credito += cant * Gold.multiplicador;
        }

>> Platinium extends Membresia
        recargarCredito(cant) {
                this.credito += min(cant * this.multiplicador, Platinium.topeMax)
        }
```

Requerimiento 6

Permitir la visualización de las partidas de cada jugador.

Consideraciones importantes:

- Nuestro Sistema se entera de las partidas cuando el Software de los juegos llaman, por API REST, a nuestro Sistema.
- El enunciado nos muestra exactamente el JSON que recibimos con los datos de la partida jugada, y es ahí cuando debemos crear la instancia de la clase Partida: no antes, ni después.

Abstracciones necesarias:

- Clase ***"Partida":*** debe contener, mínimamente:
  - Una referencia al jugador (no el id)
  - Fecha y hora
  - Duración en segundos
  - Una referencia al juego (no el id)
  - Una referencia a la máquina (no el id)

Registrar una partida que llega de una máquina

```
>> PartidasController
        registrarPartida(dataPartida) {
                Partida unaPartida = new Partida();

                Juego juegoJugado = this.repoJuego.find(dataPartida.juego_id);
                Jugador jugador = this.repoJugadores.find(dataPartida.jugador_id);
                Maquina maquina = this.repoMaquinas.find(dataPartida.maquina_id);


                unaPartida.setJuego(juegoJugado);
                unaPartida.setFechaHora(dataPartida.fecha_hora);
                unaPartida.setMaquina(maquina);
                unaPartida.setDuracionSegs(dataPartida.duracion_segundos);
                unaPartida.setPuntaje(dataPartida.puntaje);

                jugador.agregarPartida(unaPartida);

                this.repoPartidas.save(unaPartida);
        }
```

Registrar una partida que llega de una máquina

```
>> Jugador
        agregarPartida(partida) {
                if(!this.membresiaActual().tenesCreditos(partida.juego.creditos())) {
                        throw new CreditoInsuficienteException();
                }
                this.membresiaActual().descontarCreditos(partida.juego.creditos(), partida.juego);
                this.partidas.add(partida);
        }
```

Registrar una partida que llega de una máquina

```
>> Membresia
        descontarCreditos(cant, juego) {
                this.creditos -= cant - this.tipo.descuentos.map(d -> d.aplicarSobre(cant)).sum();
        }

>> DescuentoPorcentual implements Descuento
        aplicarSobre(cant, juego) {
                return (cant * this.multiplicador)
        }

>> DescuentoFijo implements Descuento
        aplicarSobre(cant, juego) {
                descuento = 0;
                if(juego.categoria == this.categoriaAplicable) {
                        descuento = cant - this.valorFijo;
                }
                return descuento;
        }
```

Requerimiento 7

Guardar registro del crédito que el jugador tiene disponible para la membresía que se encuentra activa (más allá de que la tarjeta también tenga este dato guardado).

Abstracciones necesarias:

- Clase ***“Membresía”***
  - Debería tener un atributo “activo” para saber si es la membresía activa o no.
  - Debería tener los ***créditos*** actuales en esa membresía.

Requerimiento 8

Permitir la gestión de premios y el canje de puntos por éstos.

Abstracciones necesarias:

- Clase ***"Premio"***: Debería tener como atributo, mínimamente:
  - Un nombre
  - Puntos que cuesta

- ***Logística***: Podría considerarse la utilización del patrón Adapter para llamar al servicio externo. De ser así, Logística podría delegar en LogisticaEnviosYa.

Canjear premio sea en un establecimiento o por internet

```
>> Jugador
        canjear(premio) {
                if(this.puntos < premio.costoEnPuntos()) {
                        throw new PuntosInsuficientesException();
                }
                this.descontarPuntos(premio.pcostoEnPuntos());
                this.premios.add(premio);
        }

        canjear(premio, datosEnvio) {
                this.canjear(premio);
                ServiceLocator.getInstance().get(Logistica.class).despachar(premio, datosEnvio)
        }

>> LogisticaEnviosYa implements LogisticaAdapter
        despachar(premio, datosEnvio) {
                body = this.generarJson(datosEnvio);
                this.httpClient.post(this.url, body);
        }
```

Requerimiento 9

Calcular la cantidad de puntos que hizo un jugador en un mes para permitir la generación del reporte mensual con los mejores 100 jugadores.

Consideraciones necesarias:

- El requerimiento, puntualmente, decía *"(…) Mensualmente se debe generar un reporte automático con los mejores 100 jugadores (…)"*. Para que el reporte se genere de forma automática necesitamos generar un Cron Task que ejecute, cada cierto tiempo, y se encargue de prepararlo y enviarlo. Esto sirve para el punto 4).

- Es necesario agregar un método "***puntosDelMesAnio***" en el Jugador.

Gracias