Nombre: Samuel
Apellidos: Alarco Cantos
Correo Electrónico: s.alarco@outlook.com
Teléfono de Contacto: +353 831317171

PRÁCTICA DE PROGRAMACIÓN ORIENTADA A OBJETOS – CURSO 2018 / 2019

Contenidos

Introduccion

Se va a implementar parte de la funcionalidad de un programa para gestionar un parque de atracciones. La memoria esta dividida por tareas, implementando el código paulatinamente. Se hace notar que el programa completo y funcional, con interfaz de menus de texto incluido para probar dicho programa, no estará disponible hasta la Tarea 4, que reúne todo el código implementado en las otras tareas. Asi, para inspeccionar el código fuente de todas las funcionalidades en su version mas completa, se recomienda revisar el código implementado en la carpeta Tarea 4

En el Anexo I se encuentra información sobre las clases individuales, y en el Anexo II copias del código fuente.

Tarea 1: Modelo General del Parque

Antes de empezar a implementar un modelo del parque de atracciones, se debe plantear una jerarquía de clases que permita modelar de la manera más general posible cualquier funcionamiento del parque, intentando evitar restricciones que contradigan lo especificado en el enunciado. Se debe modelar especialmente el funcionamiento de las entradas del parque, los trabajadores empleados en el parque, y las atracciones del parque. El diseño planteado demuestra varias técnicas propias de la programación orientada a objetos e intenta usar al máximo propiedades como la herencia para facilitar el mantenimiento y reutilización del código. En este apartado se esboza esta primera jerarquía de clases, junto con detalles sobre ciertas interfaces que puedan ser necesarias. Muchas de las implementaciones concretas de las clases serán expuestas con más claridad en los apartados que siguen, y en los Anexos I y II. También se discutirán ciertas decisiones de diseño y su relevancia para la prueba.

Se hace notar que el código fuente de la carpeta TAREA_1 es solo para dar un ejemplo de la estructura y posible implementacion de estas clases. Las clases no serán implementadas y parametrizadas debidamente hasta tareas posteriores.

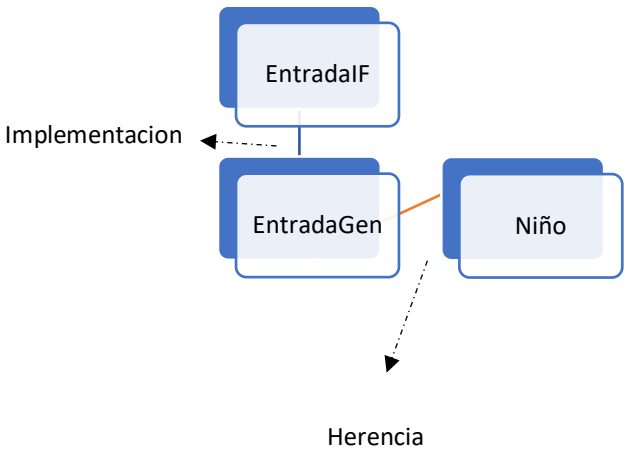
Visitantes y Entradas

Para simplificar el diseño y hacerlo lo más eficiente posible, se ha abstraído el concepto de visitante al concepto de entrada. Un visitante es su entrada, y dentro del programa no hay diferencia entre el visitante y su entrada. Así, cualquier visitante está totalmente caracterizado por su entrada, y se almacenara en las estructuras del programa del parque según la entrada que posea. Esto es así porque en el proceso de compra de entrada se debería obtener toda la información relevante del visitante, para uso posterior.

Dado que todas las entradas comparten una funcionalidad propia, se ha diseñado una interfaz de entrada, EntradaIF. Esta interfaz será luego implementada por dos clases de entrada, una clase para todas las entradas (EntradaGen) excepto la de Niños, y otra implementación específica para las entradas de niños. Esto es así porque la entrada para niños, como se verá más tarde, requiere funcionalidad esta, ya que los niños necesitan un acompañante para tener acceso al parque. Esto es algo que las demás entradas no necesitan. El uso de una interfaz común facilita que en el futuro se puedan incluir nuevos tipos de entradas, como surjan las necesidades del parque. En la interfaz se incluyen las siguientes funcionalidades:

- Obtener precio de entrada
- Obtener fecha de compra
- Indicar si la entrada es VIP
- Indicar si la entrada es familiar
- Obtener temporada de la entrada
- Obtener a qué tipo de visitante pertenece la entrada y que descuentos se han aplicado
- Obtener edad del visitante
- Aplicar descuentos a la entrada

Las relaciones de herencia entre las clases serán las siguientes:



Las clases EntradaGen y Niño implementan la interfaz EntradaIF. Además, la clase Niño **hereda** toda la funcionalidad de la clase EntradaGen, añadiendo la posibilidad de incluir un acompañante a la entrada

En los diagramas, naranja indica herencia, azul implementación de una interfaz, y verde uso de una clase.

La interfaz tendría un aspecto similar a este:

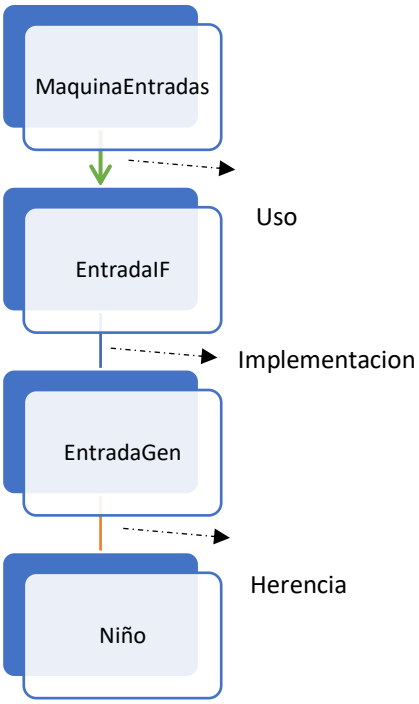
```
/**
 * Interfaz principal de las entradas, con
 * toda la funcionalidad necesaria.
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */
import java.time.*;

public interface EntradaIF
{
    float getPrecio();

    LocalDate getDate();

    boolean getVIP();

    void setVIP();

    String getTemporada();

    void setTemporada();

    boolean getFamilia();
```

```
void setFamilia();

void applyDescuento(String tipo, float descuento);

String getTipo();

int getEdad();
}
```

Se observa que la gran parte de los métodos de la interfaz son métodos getter y setter para los atributos ya mencionados arriba. Esto se explicará mejor en la implementación que se hará en la Tarea 2.

Como nota de decisión de diseño, se hace notar que los descuentos caracterizan en la mayor parte de los casos el tipo de entrada. Así, una entrada senior es tal porque tendrá un descuento senior, y una entrada de desempleado es tal porque tiene un descuento de desempleado. Esto se ha hecho así para simplificar el código y por el hecho de que la mayor diferencia entre los diferentes tipos de entrada son los descuentos aplicados. Estos se almacenarán dentro de la entrada, posiblemente en un HashMap, junto con los valores de dichos descuentos. Esto permite que los descuentos sean acumulables, y que puedan ser registrados para uso posterior en el sistema del parque.

Para facilitar y gestionar la creación de nuevas entradas, se diseñará una clase MaquinaEntradas, que se encargará de los detalles de la creación y configuración de nuevos objetos que usen la interfaz EntradaIF. Esto promueve un diseño dirigido por responsabilidad, y hace posible que el resto del programa no tenga que preocuparse de los pormenores de la creación de entradas, ya que siempre podrá usar la MaquinaEntradas. Así, la jerarquía quedaría de la siguiente manera, donde la MaquinaEntradas hace uso directo de los métodos de EntradaIF.



Atracciones

Dado que todas las atracciones del parque compartirán una serie de métodos comunes, también se ha usado una interfaz general para las atracciones del parque: AtraccionIF. Esta interfaz debe ofrecer las siguientes funcionalidades:

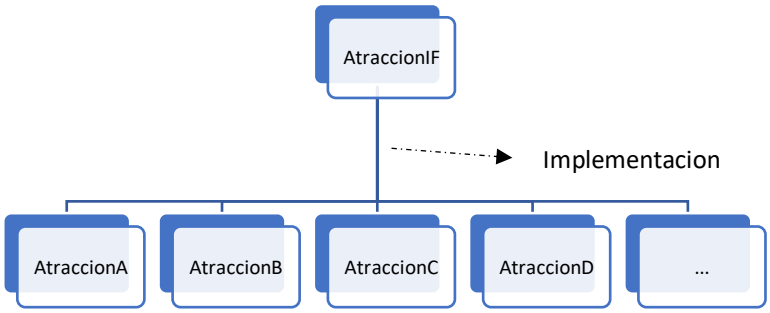
- Obtener tipo de atracción
- Indicar si atracción posee acceso VIP
- Obtener altura mínima para acceder
- Obtener altura máxima para acceder
- Obtener edad mínima para acceder
- Indicar si niños tienen acceso
- Indicar si adultos tienen acceso
- Indicar cuantos trabajadores de cada tipo hacen falta para el funcionamiento de la atracción
- Almacenar los datos de los trabajadores que trabajan en la atracción
- Almacenar los datos de los visitantes que usen la atracción

Así, la interfaz AtraccionIF contendría las siguientes cabeceras:

```
/**
 * Write a description of class Atraccion here.
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */

import java.util.List;
import java.util.LinkedList;
public interface AtraccionIF
{


    /**
     * Constructor for objects of class Atraccion
     */

    public String getTipo();

    public boolean getVIP();

    public void setVIP(boolean VIP);

    public float getMinAlturaCM();

    public void setMinAlturaCM(float max);

    public float getMaxAlturaCM();
```

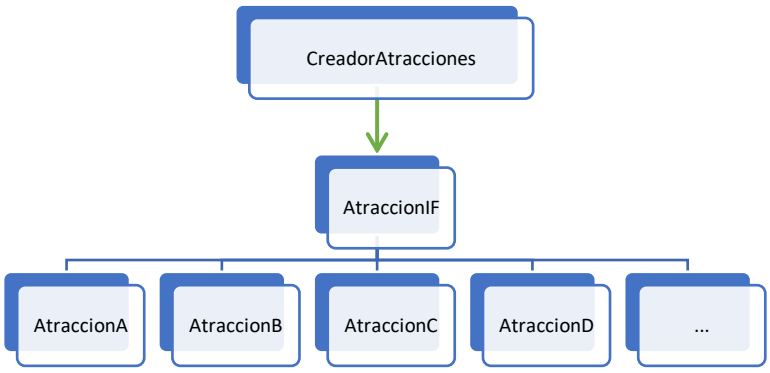
```
    public void setMaxAlturaCM(float max);

    public int getEdad();

    public void setMinEdad(int min);

    public boolean getAccesoNiños();

    public void setAccesoNiños(boolean acceso);

    public boolean getAccesoAdultos();

    public void setAccesoAdultos(boolean acceso);

    public void setNumRespAtracc(int num);

    public int getNumRespAtracc();

    public void setNumAyuAtracc(int num);

    public int getNumAyuAtracc();

    public void addTrabajador(Trabajador trabajador);

    public List getTrabajadores();

    public void usar(EntradaIF entrada);

    public List getUsuarios();

}
```

En las Tarea 2, se diseñarán varias clases que implementen esta interfaz, ya que se prevé que haya varios tipos de atracción. El uso de clases separadas por cada tipo atracción hace posible que aparte de la funcionalidad general que todas las atracciones deben tener (implementar la interfaz de atracción), los diversos tipos de atracciones pueden tener funcionalidad adicional especifica al tipo. Así, la jerarquía de clases tendría el siguiente aspecto:



Donde se ve claramente que las clases AtraccionA, AtraccionB etc implementan la interfaz AtraccionIF.

Por las mismas razones que para las clases de entrada, se ha añadido en el diseño una clase CreadorAtracciones, que cumple la misma función que MaquinaEntradas, esta vez para los objetos Atraccion…



Trabajadores del Parque

Se deben diseñar estructuras para modelar el personal del parque. Aunque haya varios tipos de trabajador, toda clase trabajador debe ofrecer una funcionalidad común, incluyendo:

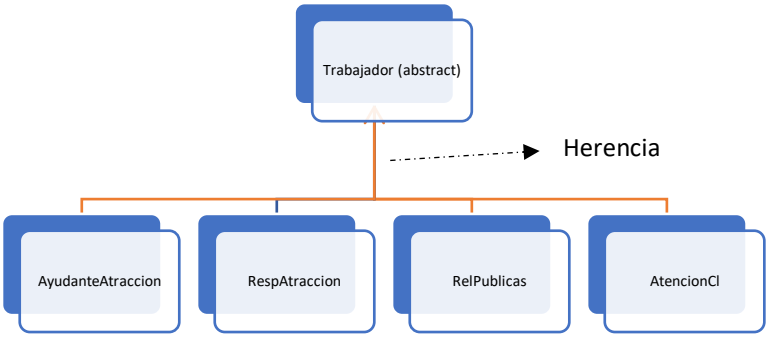
- Obtener lista de atracciones de las que se encarga el trabajador
- Modificar dicha lista de atracciones
- Obtener sueldo del trabajador
- Obtener tipo de trabajador

En esta práctica, no se usa la funcionalidad que ofrece la lista de atracciones de las que se encarga el trabajador, pero esto puede resultar útil para revisiones futuras. Dado que todo trabajador debe mantener una lista de las atracciones de las que se ocupa (si es que existen), se ha decidido usar una clase abstracta en vez de una interfaz para generalizar en concepto de trabajador, ya que en las clases abstractas se pueden incluir atributos, no así en las interfaces.

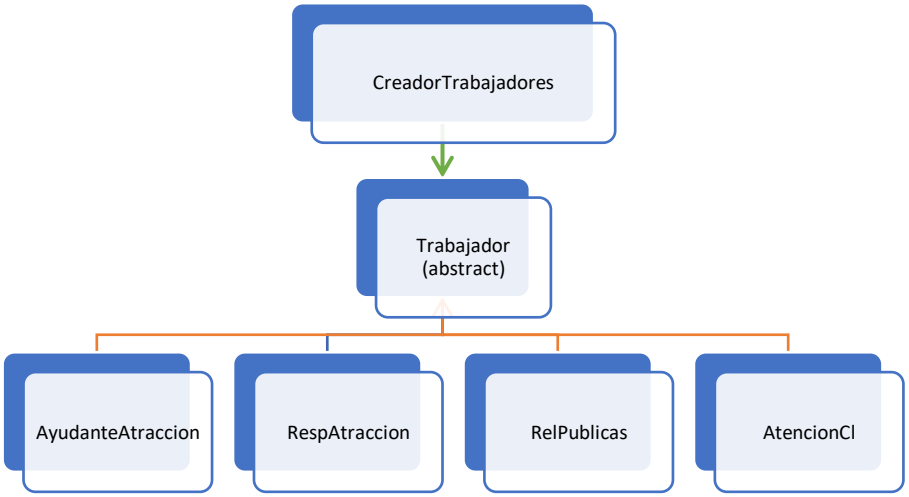
```
/**
 * Clase abstracta Trabajador con metodos
 * comunes a todos los trabajadores.
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */

import java.util.List;
import java.util.LinkedList;

public abstract class Trabajador
{
   private List<AtraccionIF> atracciones;
```

```
    public Trabajador()
    {
        atracciones = new LinkedList<AtraccionIF>();
    }

    abstract public TiposTrabajadores getTipo();

    public List<AtraccionIF> getAtracciones()
    {
        return atracciones;
    }

    public void setAtracciones(LinkedList<AtraccionIF> atracciones)
    {
        this.atracciones = new LinkedList(atracciones);
    }

    public void addAtraccion(AtraccionIF atraccion)
    {
        atracciones.add(atraccion);
    }

    public void removeAtraccion(AtraccionIF atraccion)
    {
        atracciones.remove(atraccion);
    }

    public void clearAtracciones()
    {
        atracciones.clear();
    }

    public int numeroDeAtracciones()
    {
        return atracciones.size();
    }

    abstract public float getSueldo();
}
```

Como se puede observar, ya se ha incluido cierta funcionalidad. Esto será implementado y explicado con más detalle en el apartado de la Tarea 2. Se ha intentado utilizar al máximo la encapsulación de los atributos de las clases, como es buena práctica en la programación orientada a objetos. En la Tarea 2 se implementarán diferentes tipos de trabajadores que extenderán la clase abstracta Trabajador. Cada una deberá implementar el método getSueldo() y el método getTipo(), y heredan (mostrado en naranja), los métodos de la clase abstracta Trabajador.



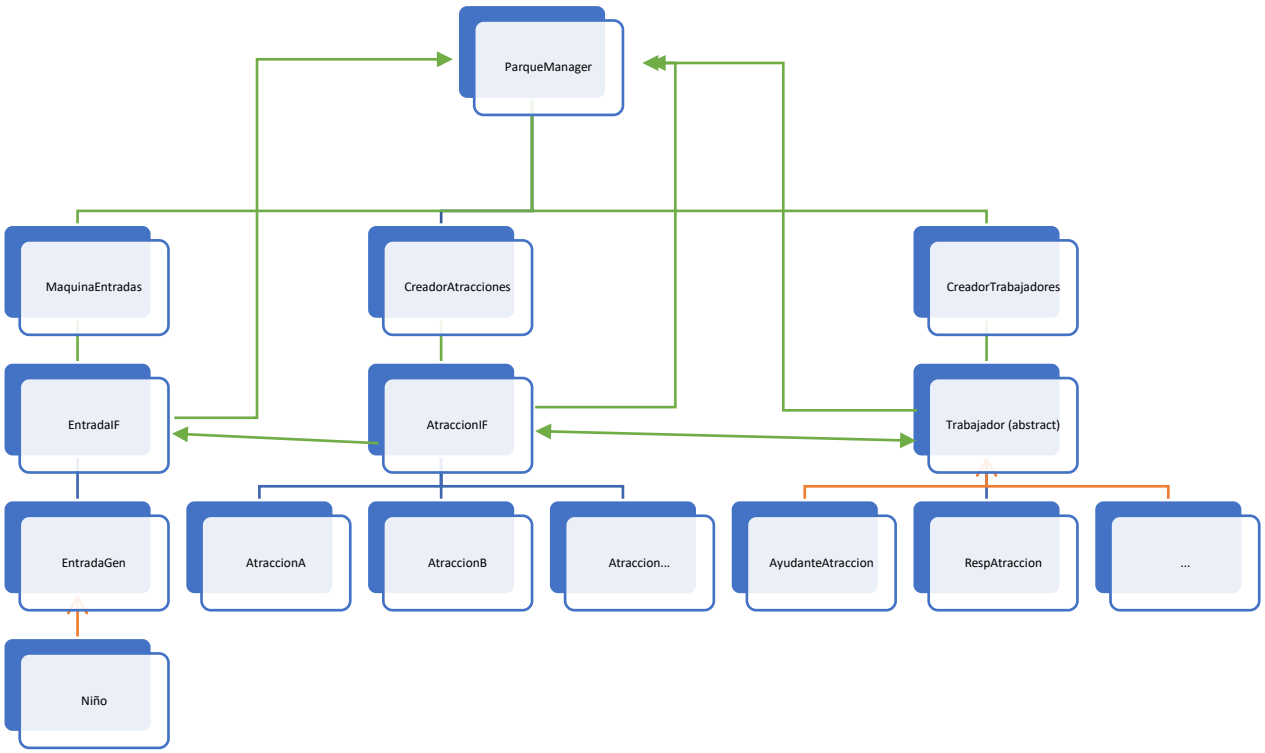
Por último, al igual que para las entradas y las atracciones, también se ha diseñado una clase CreadorTrabajadores.



Para facilitar la implementación del programa en otros idiomas, se ha decidido identificar a los diferentes tipos de trabajadores utilizando una clase enum TiposTrabajadores, que contiene todos los tipos de trabajadores. Su implementación se discute en el apartado de la Tarea 2.

ParqueManager

Por último, se quiere diseñar una clase que contenga las estructuras necesarias para almacenar y gestionar los objetos y funcionalidades principales del parque. Esta clase hará uso de objetos MaquinaEntradas, CreadorAtracciones y CreadorTrabajadores para poder crear nuevos objetos de los respectivos tipos, y usará listas (List) para almacenar las entradas, las atracciones y los trabajadores creados. En las Tareas 3 y 4 también se encargará de almacenar objetos como el analizador estadístico, y de proveer métodos que faciliten ciertas demostraciones del programa. Así, un diagrama unificado de las relaciones entre las clases podría ser el siguiente:



Como se puede observar, el ParqueManager también depende las interfaces de las entradas, atracciones, y trabajadores, ya que tiene que almacenar dichos objetos en sus listas.

Se vuelve a indicar que, en el diagrama,línea azul indica implementación de una interfaz, línea naranja indica herencia de una clase del nivel superior, y línea verde uso de una clase (flecha apunta a la clase usada por la clase usante)

Clases de Apoyo y Funcionalidad Añadida

Para permitir que el modelo del parque sea los más general posible, se han pensado varias clases de apoyo que permitirán parametrizar en iteraciones posteriores. Estas clases serán expuestas con más detalle en las Tareas sucesivas, pero damos un pequeño esbozo de su posible uso en esta sección.

Clase PARAMETROS

Esta clase contendrá atributos estáticos que podrán ser usados por todo el programa. Un uso posible que se prevé es contener datos para configurar las entradas, tales como el precio base, los descuentos o incrementos de entradas familiares y VIP, sueldo base de los trabajadores etc. Esto permite parametrizar fácilmente el programa para un uso concreto. Estos valores podrían también ser contenidos en las clases que las usan, pero contener estos valores en una misma clase y de forma estática permite una configuración rápida del programa, y puede facilitar el mantenimiento. Se darán más detalles sobre esta clase en la Tarea 2.

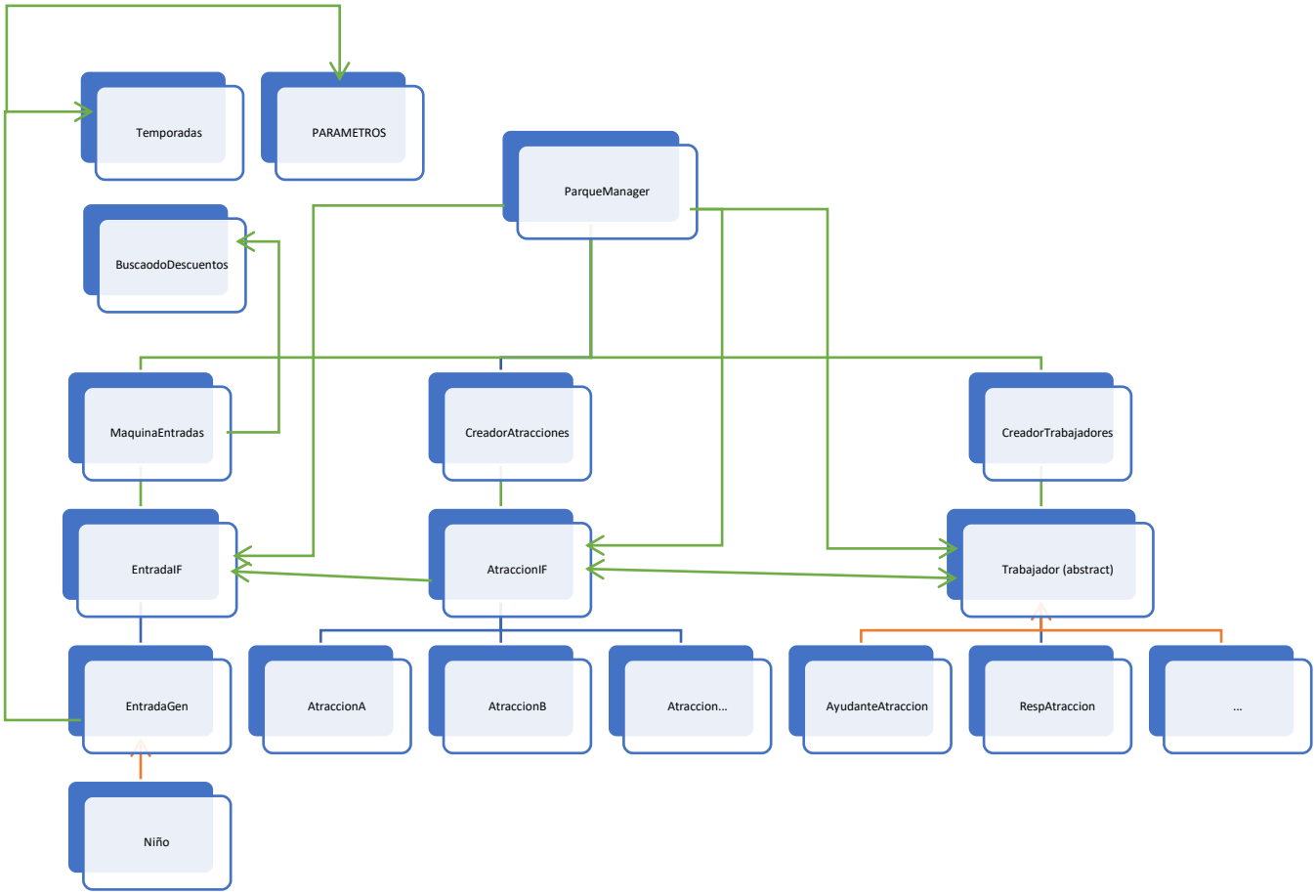
Clase BuscadorDescuentos

Otra clase de apoyo será la clase BuscadorDescuentos. Esta clase contendrá los descuentos posibles y acumulables que se puedan aplicar a las entradas, y tendrá funcionalidad para poder buscar dichos descuentos según una referencia, como su nombre. Esto favorece un diseño dirigido por responsabilidad. Se darán más detalles sobre esta clase en la Tarea 2.

Clase Temporadas

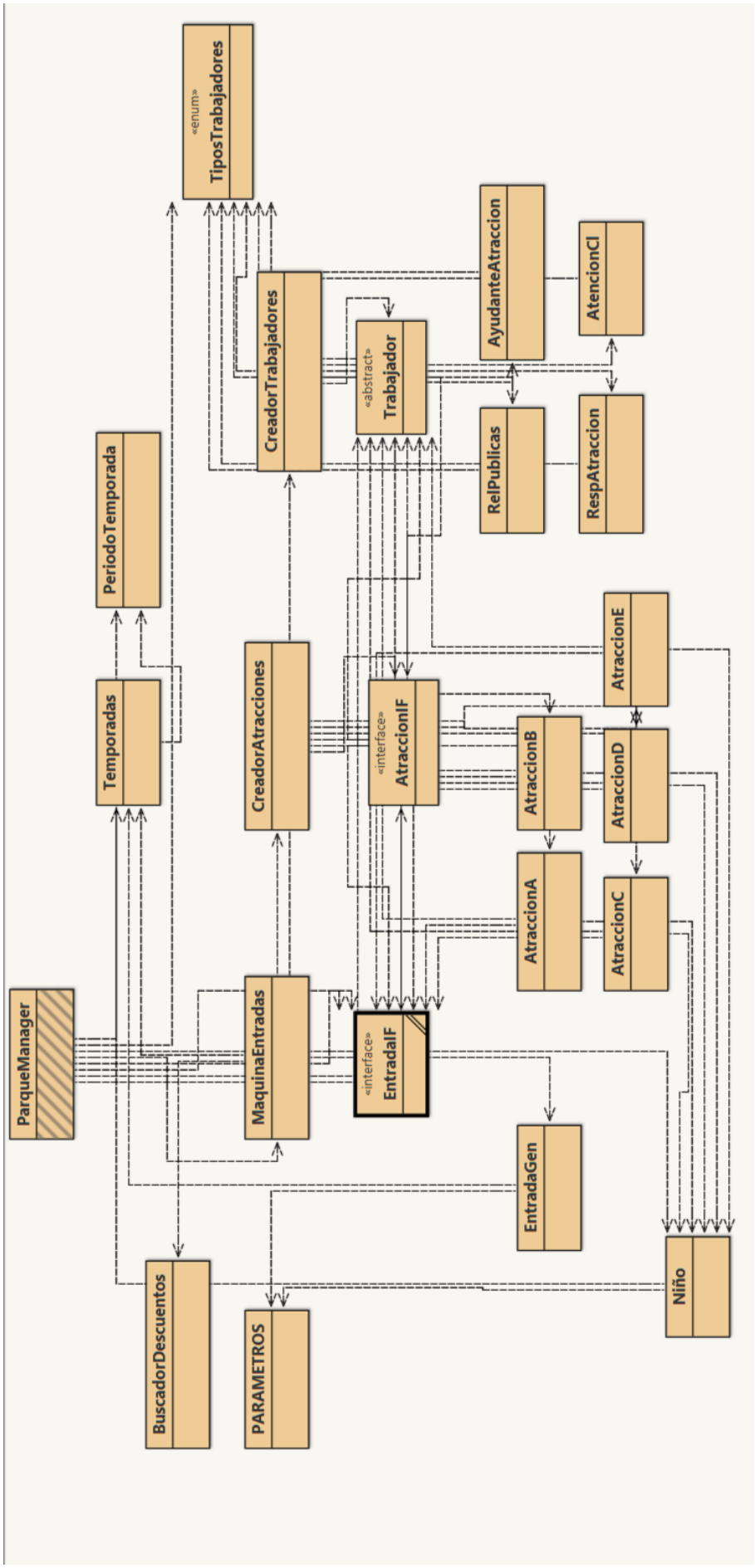
De manera similar a la clase BuscadorDescuentos, la clase Temporadas se encargar de almacenar y proveer datos sobre las diversas temporadas de funcionamiento del parque, con los descuentos o subidas de precio asociadas. Así se separa esta información de la implementación de las entradas, disminuyendo el acoplamiento y favoreciendo un diseño dirigido por responsabilidad.

Un diagrama más completo con los posibles usos de estas clases sería el siguiente:



Se vuelve a indicar que, en el diagrama, línea azul indica implementación de una interfaz, línea naranja indica herencia de una clase del nivel superior, y linea verde indica uso de una clase (flecha apunta a la clase usada por la clase usante).

El diagrama completo de los **usos** después de una implementación más exhaustiva de la clase se muestra en la página siguiente, tal y como es generado por el entorno BlueJ:


«enum»
TiposTrabajadores
AyudanteAtraccion
AtencionCl
CreadorTrabajadores
«abstract»
Trabajador
RelPublicas
RespAtraccion
PeriodoTemporada
Temporadas
CreadorAtracciones
«interface»
AtraccionIF
AtraccionE
AtraccionB
AtraccionD
AtraccionA
AtraccionC
ParqueManager
MaquinaEntradas
«interface»
EntradaIF
EntradaGen
BuscadorDescuentos
PARAMETROS
Niño

Tarea 2: Concretización de un Parque

Una vez que se ha modelado el parque (Tarea 1) de la manera más general posible, en la tarea 2 concretizamos un ejemplo de parque añadiendo valores concretos para los precios, descuentos, tipos de entrada, y estructuras como atracciones etc.

Explicaremos este proceso de concretización por partes, para añadir claridad a la memoria de la práctica y facilitar la lectura. Es importante hacer notar que donde el enunciado pide un numero especifico de atracciones, trabajadores etc, el menú del programa ofrece opciones para generar automáticamente este contenido. Por ejemplo, para generar automáticamente el numero de atracciones especificado por la Tarea 2 del enunciado basta con acceder al Menu Atracciones desde el menú principal y usar la opción para generar atracciones automáticamente.

Clase PARAMETROS:

Como ya se mencionó en la Tarea 1, se diseñó una clase PARAMETROS con la sola función de almacenar ciertos valores estáticos que puedan ser accedidos por todas las clases. Esto facilita la labor de concretizar o modificar estos valores. En la práctica, solo se ha usado como ejemplo para almacenar valores concernientes a los precios de las entradas y algunos descuentos generales aplicables, como el descuento Familia, que se puede aplicar a cualquier entrada. Pero en el futuro esta clase se puede usar para parametrizar cualquier otro valor de uso general. Esto facilita el mantenimiento y la comprensión del código del sistema.

Los valores específicos del enunciado son los siguientes:

```
public static float PRECIO_BASE = 60f;
public static float DESCUENTO_NIÑO = 50f;
public static float PRECIO_VIP_BASE = 50f;
public static float DESCUENTO_FAMILIA = 10f;
```

Los descuentos están expresados en porcentajes del precio al que se descuentan.

Clase BuscadorDescuentos:

Para los descuentos específicos a cada entrada, se ha usado decidido usar un método más dinámico, a través de una clase objeto *BuscadorDescuentos*. En líneas generales, este objeto utiliza una estructura HashMap para almacenar los porcentajes de cada descuento aplicable, utilizando el nombre del descuento como clave (un String). Esta clase se encarga de almacenar y buscar dichos descuentos cuando otra funcionalidad así lo pida. También permite añadir nuevos descuentos para uso posterior dinámicamente con el método *public void addDescuento(String key, float descuento).* Los descuentos especificados en el enunciado se han añadido al constructor para demostrar en funcionamiento de la clase más fácilmente:

```
public BuscadorDescuentos()
    {
        // initialise instance variables
        mapaDescuentos = new HashMap<String, Float>();
        mapaDescuentos.put("carnet joven", 10f);
        mapaDescuentos.put("discapacitado", 20f);
        mapaDescuentos.put("estudiante", 10f);
        mapaDescuentos.put("veterano", 10f);
        mapaDescuentos.put("senior", 35f);
    }
```

Para más detalles sobre la implementación y diseño de esta clase consultar el Anexo I y II.

Clase Temporadas y PeriodoTemporadas

Se ha diseñado una clase Temporadas para almacenar la información sobre las diferentes temporadas del parque, con sus respectivos descuentos. Esto de nuevo se hace para promover un diseño dirigido por responsabilidad. En línea general, esta clase hace uso de dos estructuras HashMap. Uno almacena los descuentos para cada temporada, usando como clave el nombre de la temporada (String) y como valor asociado el descuento (porcentaje) de la temporada. El otro HashMap almacena los periodos de las diferentes temporadas. Esta clase ofrece funcionalidad sea para buscar a que temporada pertenece una fecha, o que descuento se debe aplicar dependiendo de la temporada.

Según lo especificado en el enunciado, se ha incluido en el constructor de la clase Temporadas los periodos de algunas temporadas:

```
  public Temporadas()
    {
        mapaDescuentosTemporadas = new HashMap<String, Float>();
        mapaDescuentosTemporadas.put("tempAlta", 115f);
        mapaDescuentosTemporadas.put("tempBaja", 85f);
        mapaDescuentosTemporadas.put("tempMedia", 100f);

        mapaTemporadas = new HashMap<PeriodoTemporada, String>();
        mapaTemporadas.put(new PeriodoTemporada(LocalDate.of(2019, Month.JANUARY,
1), LocalDate.of(2019, Month.JANUARY, 8)), "tempAlta");
        mapaTemporadas.put(new PeriodoTemporada(LocalDate.of(2019, Month.APRIL,
15), LocalDate.of(2019, Month.APRIL, 21)), "tempAlta");
        mapaTemporadas.put(new PeriodoTemporada(LocalDate.of(2019, Month.APRIL,
1), LocalDate.of(2019, Month.APRIL, 30)), "tempAlta");
        mapaTemporadas.put(new PeriodoTemporada(LocalDate.of(2019, Month.AUGUST,
1), LocalDate.of(2019, Month.AUGUST, 31)), "tempAlta");
        mapaTemporadas.put(new PeriodoTemporada(LocalDate.of(2019, Month.DECEMBER,
1), LocalDate.of(2019, Month.DECEMBER, 31)), "tempAlta");
        mapaTemporadas.put(new PeriodoTemporada(LocalDate.of(2019, Month.FEBRUARY,
1), LocalDate.of(2019, Month.FEBRUARY, 28)), "tempBaja");
        mapaTemporadas.put(new PeriodoTemporada(LocalDate.of(2019, Month.NOVEMBER,
1), LocalDate.of(2019, Month.NOVEMBER, 30)), "tempBaja");
    }
```

Para más detalles sobre la implementación y diseño de esta clase consultar el Anexo I y II.

La clase Temporadas hace uso de la clase PeriodoTemporadas. Esta es una clase diseñada para facilitar el almacenamiento de periodos entre dos objetos LocalDate. Ofrece funcionalidades como verificar si una cierta fecha se encuentra incluida en el periodo, o si dos periodos son iguales. Se usa mucho en la implementación de la clase Temporadas, y facilita el diseño de esta.

Para más detalles sobre la implementación y diseño de esta clase consultar el Anexo I y II.

Implementación de las clases diseñadas en la Tarea 1 y clases de Apoyo:

En este apartado también se implementa el código de las clases ya organizadas en la tarea 1, junto con clases para demonstrar el funcionamiento del programa generando contenido para el parque de atracciones.

Los detalles serán descritos con más extensión en el Anexo II, donde se describen los detalles de las clases. Aquí me limitare a dar algunas notas sobre decisiones de diseño e implementación que podrían resultar importantes a la hora de entender el funcionamiento del programa.

Clase EntradaGen

La mayor parte de la implementación de la interfaz EntradaIF se realiza en la clase EntradaGen (Entrada General). Esta será la clase usada para todas las entradas excepto la de niños, que requiere de funcionalidad especial.

Se ha decidido incluir atributos separados para el estado VIP y para el descuento de familia, ya que estos son descuentos con condiciones diferentes de los demás y que se pueden aplicar a cualquier entrada. Por lo tanto, no se incluyen entre los otros descuentos, y tienen métodos getter y setter propios.

Como se ha mencionado en el apartado de la tarea 1, el tipo de entrada es igual al descuento o descuentos aplicados. Así, una entrada para desempleado es tal porque se le ha aplicado un descuento para desempleado. Por lo tanto, no hay diferencia entre el tipo de entrada y el descuento aplicado. Una entrada senior es tal porque se le ha aplicado un descuento senior, y así con las varias combinaciones de descuentos existentes. Los descuentos se aplican usando el método *applyDescuento(…)*, que cumple la doble función de modificar el precio de la entrada y almacenar el descuento aplicado en un HashMap, para uso posterior. También se encarga de verificar que el total de descuentos aplicados no sobrepase el limite especificado en el enunciado de la práctica.

Clase Niño

Esta clase extiende a la clase EntradaGen, heredando sus métodos y atributos. Además, añade el atributo acompañante, ya que todo niño debe ser acompañado de un adulto. Esta entrada

debe ser de tipo EntradaGen, ya que un niño no puede ser acompañado por otro niño. La clase que crea la entrada niño se encargara de verificar que la entrada acompañante siempre sea una entrada de adulto.

Clase MaquinaEntradas

Para administrar la creación y configuración de entradas, se ha diseñado una clase de apoyo denominada *MaquinaEntradas*, la cual se ocupará de crear nuevas entradas a partir de la información y requisitos suministrados. Esto disminuye el acoplamiento, ya que las otras clases no tienen que preocuparse de cómo crear nuevas entradas, y solo deben suministrar la información al objeto *MaquinaEntradas* que se ocupara de ofrecer la funcionalidad para crear dichas entradas. Esto también facilita el mantenimiento, ya que, si se cambia la implementación de las entradas, tendrá su mayor efecto en la clase *MaquinaEntradas*, y su efecto en el resto del código será mínimo.

La clase MaquinaEntradas ofrece dos métodos públicos esenciales: nuevaEntrada(…) y nuevaEntradaNiño(…). Hace falta un método a parte para crear las entradas para niños, ya que además de toda la información necesaria para generar una entrada normal se debe además añadir un acompañante a la entrada.

Como nota sobre la implementación, se hace notar que la clase usa automáticamente la última EntradaGen registrada como acompañante para los niños. Por lo tanto, a la hora de comprar entradas, primero debería comprar la entrada el adulto que acompaña a los niños, y después los niños que serán acompañados por el adulto.

Además, la clase genera un objeto BuscadorDescuentos, que se usara para buscar los descuentos aplicables y aplicar dichos descuentos a las entradas creadas cuando así se pida. También se genera un objeto Temporadas, que se pasara en el constructor de cada entrada para que estas lo usen para determinar a qué temporada pertenecen.

Mas detalles sobre la implementación y funcionalidad de esta clase se pueden encontrar en el Anexo I y II.

Implementaciones de la Interfaz AtraccionIF

En el enunciado se especifican varios tipos de atracciones. Todos estos tipos implementan la interfaz AtraccionIF, cambiando algunos detalles dependiendo del tipo de atracción. Las otras clases solo trataran con la interfaz, lo cual aumenta la abstracción y facilita la labor de mantenimiento y la limpieza del código. Aunque en lo que concierne el enunciado de la práctica, las diferentes atracciones no cambian mucho en su funcionamiento, se ha optado por crear una clase por atracción para facilitar expansiones posibles en el futuro. Así, si en una versión futura hiciera falta modificar o añadir funcionamientos específicos a un tipo de atracción, se esto se haría con relativa facilidad.

Como nota de implementación, se hace notar que cada atracción cuenta con una lista de los visitantes que han usado la atracción, y otra con los trabajadores de la atracción. Esto

facilitara en tareas posteriores la generación de estadísticas sobre el uso de las atracciones y el coste de funcionamiento.

Mas detalles sobre la implementación y funcionalidad de esta clase se pueden encontrar en el Anexo I y II.

Clase CreadorAtracciones

De forma similar a la clase *MaquinaEntradas*, esta clase facilita la tarea de la creación de nuevas entradas. Esto promueve un diseño dirigido por responsabilidad, pero también añade un nivel de abstracción al sistema, ya que solo la clase *CreadorAtracciones* debe preocuparse de los detalles sobre la producción y configuración de nuevas atracciones. Esto disminuye el acoplamiento y facilita el mantenimiento posterior, aparte de ser una solución de diseño elegante.

La implementación de la clase es bastante simple. El método principal usa un condicional *switch* para decidir qué tipo de atracción crear, a partir del parámetro de entrada.

Mas detalles sobre la implementación y funcionalidad de esta clase se pueden encontrar en el Anexo I y II.

Extensiones de la Clase Abstracta Trabajadores

En la tarea 1 se definió la clase abstracta de Trabajadores, con los métodos y atributos comunes a todos los trabajadores del parque. Usando las especificaciones del enunciado, estamos en grado de diseñar clases específicas que extiendan a la clase abstracta. Así, se han creado las siguientes clases:

- AyudanteAtraccion (Ayudantes de Atraccion)
- RespAtraccion (Responsables de Atraccion)
- AtencionCl (Atencion al Cliente)
- RelPublicas (Relaciones Publicas)

Estas clases especificas implementan dos métodos: getTipo() y getSueldo, que son propios a cada tipo de trabajador, y por lo tanto se han encapsulado en la clase de cada tipo de trabajador. Esto disminuye el acoplamiento y es una buena manera de organizar el código.

Mas detalles sobre la implementación y funcionalidad de estas clases se pueden encontrar en el Anexo I y II.

CreadorTrabajadores

De igual manera que la clase CreadorAtracciones, esta clase se encarga de la creación de nuevos trabajadores. Esto promueve un diseño dirigido por responsabilidad, pero también añade un nivel de abstracción al sistema.

La implementación de la clase es similar a la de la clase *CreadorAtracciones*. El método principal usa un condicional *switch* para decidir qué tipo de atracción crear, a partir del parámetro de entrada.

Mas detalles sobre la implementación y funcionalidad de esta clase se pueden encontrar en el Anexo I y II.

TiposTrabajadores (enum)

La clase TiposTrabajadores es una clase enum con los identificadores de los diferentes tipos de trabajadores. Estos identificadores se usan en todo el programa para identificar los diferentes tipos de trabajadores. El uso de este método elimina el uso de cadenas de texto que pueden ser especificas a un idioma. Así, el programa se puede implementar en diferentes idiomas fácilmente.

En revisiones posteriores, el uso de estas estructuras se podría usar en otros sitios del programa, pero por falta de tiempo esto no ha sido posible.

Detalles sobre la implementación de esta clase se pueden encontrar en el Anexo II.

Clase ParqueManager

Por último, esta parte del programa hace uso de la clase *ParqueManager*. Esta clase se encarga de almacenar y gestionar los objetos y funcionalidades principales del programa.

Para almacenar los diferentes objetos, la clase *ParqueManager* tres listas (LinkedList): para los visitantes (entradas), los trabajadores, y para las atracciones del parque. Se ha elegido usar la estructura LinkedList porque no está acotada, pero permite ordenar los elementos.

Además de estas estructuras de almacenamiento, la clase *ParqueManager* también cuenta con varios objetos para llevar a cabo sus funciones. Tiene una instancia de la *MaquinaEntradas* para crear nuevas entradas y añadirlas a la lista de entradas. También tiene un *CreadorTrabajadores* para crear nuevos trabajadores y añadirlos a la lista de trabajadores, y un *CreadorAtracciones* para crear nuevas atracciones y añadirlas a la lista de atracciones.

Para facilitar las pruebas demonstraciones de uso del sistema, la clase *ParqueManager* también cuenta con varias funciones de apoyo, usadas por ejemplo para el recuento de trabajadores existentes o para generar un uso aleatorio de las atracciones. Estas funciones se usarán sobre todo en las Tareas 3 y 4, aunque ya se mencionan en este apartado para no fraccionar demasiado la lectura.

Esta clase seria la que se conectase a una interfaz de texto o a una interfaz gráfica si en el futuro se quisiese implementar dicha interfaz.

Se hace notar que para la generación del contenido especificado en el enunciado, se hace uso de varias clases que tienen como única función la generación de dicho contenido. Estas clases se usan posteriormente a lo largo del programa, y se describen en el apartado que sigue a la descripción de la Tarea 4.

Mas detalles sobre la implementación y funcionalidad de esta clase se pueden encontrar en el Anexo I y II.

Clase AnalizadorEstadisticas para demostrar algunas funcionalidades

Para mostrar información sobre estructuras y objetos creados, y así facilitar hasta cierto punto la demostración del funcionamiento de algunos aspectos del programa, se ha dejado en la Tarea 2 una versión temprana de la clase *AnalizadorEstadisticas*. Esta clase no es implementada y discutida por completo hasta el apartado de la Tarea 3, pero se hace notar como nota final que se usan métodos de esta clase para los fines ya descritos.

Tarea 3: Generación de Estadísticas

Se ha diseñado una clase especializada para generar varios resúmenes estadísticos sobre el parque de atracciones. Esta clase será expandida en la Tarea 4. Para los requisitos de esta tarea, la clase *AnalizadorEstadisticas* genera las siguientes estadísticas:

- Un análisis del número de visitantes diarios, de cada semana, de cada mes, y del año, junto con sus respectivos promedios.
- Información sobre el precio medio de las entradas vendidas cada día, semana, mes y año.
- Visitas medias registradas a cada atracción del parque

Para poder generar dicha información, la clase debe obtener los datos almacenados en el objeto de clase *ParqueManager*. Que como ya se ha mencionado se ocupa de almacenar y administrar las diferentes estructuras del parque. Dichos datos se pasan al objeto *AnalizadorEstadisticas* a través del constructor, en forma de Listas:

```
public AnalizadorEstadisticas(List<EntradaIF> listaEntradas,
List<AtraccionIF> listaAtracc, List<Trabajador> listaTrab)
    {
        ListaEntradas = listaEntradas;
        ListaAtracciones = listaAtracc;
        ListaTrabajadores = listaTrab;
    }
```

Como en Java se manipulan objetos por referencia, se verifica que una vez creado el objeto *AnalizadorEstadisticas* las listas se mantienen actualizadas. Los métodos que se encargan de generar las diferentes estadísticas están descritos en detalle en el Anexo I, y el código fuente se puede encontrar en el Anexo II.

Se ha decidido usar una clase específica para generar estas estadísticas para promover un diseño dirigido por responsabilidad. Esto facilita la labor de mantenimiento, y hace que sea sencillo añadir más funcionalidad a la clase en actualizaciones futuras, como se hará en la tarea 4. En el diseño de los métodos para generar las varias estadísticas, se ha intentado al máximo reutilizar el código usado para otros métodos, con ligeras modificaciones dependiendo de los requisitos de cada resumen estadístico. Dado que en general los resúmenes tienen la misma estructura (agrupación por días, semanas, meses y año), se ha usado la misma estructura de bucles *for* y condicionales. Para más explicaciones, como ya se ha dicho, consultar el Anexo I y II.

Tarea 4: Clase AtraccionesFuncionando y Expansión de la Clase AnalizadorEstadisticas – PROGRAMA COMPLETO

En esta tarea se debe general estadísticas sobre los gastos de personal diario. Los sueldos base se especifican en el enunciado de la práctica. Dado que para el apartado anterior ya hemos diseñado una clase que se ocupa de la generación y cálculo de estadísticas que conciernen el uso del parque, añadiremos esta funcionalidad en dicha clase. El código fuente de la clase AnalizadorEstadisticas expandida de puede encontrar en el anexo.

Esta expansión hace uso del objeto *atraccionesActivas*, que se pasa al constructor de la clase *AnalizadorEstadisticas* cuando se crea. El objeto *atraccionesActivas* es de clase *AtraccionesFuncionando*, el código fuente de la cual se puede encontrar en el anexo. Esta clase se ocupa de almacenar y suministrar información las atracciones que están activas y en funcionamiento durante diferentes periodos del año. Utiliza un HashMap que almacena valores de tipo List<AtraccionIF> accesibles a través de sus respectivos periodos de funcionamiento, que están representados usando objetos *PeriodoTemporada*. Así se reutiliza el código de la clase *PeriodoTemporada*. De esta manera, se pueden buscar periodos dentro del HashMap, y el HashMap devolverá las atracciones activas dentro de dicho periodo. Detalles de la implementación y funcionalidad de la clase *AtraccionesFuncionando* se pueden encontrar en el anexo que trata los detalles de las clases. Lo que más cabe destacar es que dicha clase posee la función pública *getAtracciones(LocalDate fecha),* que recibe una fecha y devuelve un objeto Lista de las atracciones que se encuentran activas en dicha fecha . Esta funcionalidad se usará en la clase *AnalizadorEstadisticas*. De esta manera, tenemos una clase que se ocupa específicamente del almacenamiento y suministro de dicha información, encapsulándola y a la vez proveyendo funciones de búsqueda y validación para la correcta inserción de datos sobre los periodos de funcionamiento de cada atracción. Esto disminuye el acoplamiento y promueve un diseño dirigido por responsabilidad.

La función pública que se encarga de generar las estadísticas sobre el gasto del personal diario en la clase *AnalizadorEstadisticas* es la función *public void resumenGastoPersonal(int year).* En términos generales, la función pide al objeto *atraccionesActivas* la lista de atracciones activas para cada día del año especificado. Se empieza en el primer día del año especificado, y un bucle *for* analiza cada día del año organizando la información en una jerarquía de semanas, meses y el año, utilizando la misma estructura de condicionales que se usó en la tarea 3 para la generación de otras estadísticas. La información se va imprimiendo en pantalla a través de la consola. Para actualizar los contadores y los sumadores del año, mes y semana, se usan funciones de soporte que recorren la lista de atracciones funcionando y por cada atracción, la lista de trabajadores en esa atracción, añadiendo cada trabajador al contador (en el caso de *float resolvContador(…)*) o sumando su sueldo al sumador (en el caso de *float resolvSumador(…)*). Por último, también se hace uso de la función privada *float round(float d, int decimalPlace)* para redondear a dos números decimales.

Se ha intentado reutilizar al máximo el código desarrollado para la clase *AnalizadorEstadistica*, para que se integre lo mejor posible en su estructura y para facilitar labores de mantenimiento en el futuro.

Clase MenuInterface

La clase MenuInterface se encarga de ofrecer la funcionalidad de una interfaz de texto dirigida por menus. La clase ofrece un único metodo publico *menu1()* : el menú principal del programa. Desde este menú se puede acceder a toda la funcionalidad del programa.

Aquí se da un resumen de la jerarquía de menus

- Menu Entradas
    - Nueva Entrada Individual
        - Nueva Entrada Adulto (todas las entradas generales
        - Nueva Entrada Niño
    - Importar Entradas desde Archivo
        - Se importan entradas usando la funcionalidad de la clase GeneradorVisitantes para importar entradas desde el archivo Visitantes.txt
    - Resumen Entradas
        - Recuento de las entradas por tipo de entrada
    - Salir
- Menu Atracciones
    - Nueva Atraccion
    - Generar Atracciones (se generan las atracciones especificadas en el enunciado de la Tarea 2)
    - Activar/Desactivar Atracciones
        - Menu con funcionalidad para activar atracciones por periodo de tiempo
    - Generar uso aleatorio de las atracciones
        - Genera un uso aleatorio de las atracciones para ser usado en estadisticas
    - Resumen Atracciones
        - Recuento de las atracciones por tipo de atraccion
- Menu Trabajadores
    - Resumen de los trabajadores
        - Los trabajadores se generan automáticamente cuando se generan atracciones. Esta opción recuenta y muestra el numero de trabajadores de cada tipo en el parque
- Menu Estadisticas
    - Informacion Visitantes del Parque (numero de entradas por fecha en un año del parque)
    - Informacion Precio Entradas (estadísticas sobre el precio de las entradas con promedios diarios, semanale, mensuales y anuales)

    - Informacion Uso de Atracciones (información sobre los visitantes que usan una cierta atraccion)
    - Informacion Gastos del Personal Diario (Tarea 4) (estadísticas sobre los gastos de personal, que fluctúan según las atracciones que esten activas o no).
- Salir

Para crear un objeto *MenuInterface*, hace falta pasarle al constructor el objeto ParqueManager del parque, para que el objeto *MenuInterface* tenga disponible todas las funciones y datos almacenados del parque.

La clase *MenuInterface* tambien se ocupa en muchas ocasiones de validar la entrada de datos antes de que estos datos sean usados en las diferentes funcionalidades del programa.

Clases para Generar Contenidos

Aunque no forman parte integra del modelo del parque, y por lo tanto no se han discutido hasta ahora, se han incluido varias clases para generar contenido e importarlo en el parque. Esto facilita la demostración de algunas funcionalidades. Particularmente, tenemos la clase *GeneradorContenido* y la clase *GeneradorVisitantes*.

Clase GeneradorContenido

La clase *GeneradorContenido* se usa a lo largo del proyecto (desde la Tarea 2) para generar los contenidos del parque especificados por el enunciado (atracciones, trabajadores necesarios según el número de atracciones etc.). Contiene un único método estático *generarContenido(ParqueManager)* al que se le pasa el objeto ParqueManager, y que desencadena una serie de métodos para crear todo el contenido necesario. La versión más completa, la de la Tarea 4, se puede encontrar en el Anexo II.

La funcionalidad de esta clase se puede acceder a traves de los menús del programa, que ofrecen opciones para la generación automática de contenido.

Clase GeneradorVisitantes

Esta clase es de especial importancia a la hora de importar los visitantes del parque. Para esta práctica, los visitantes se importan desde un archivo Visitantes.txt, en la cual la información de los visitantes ha sido codificada en formato delimitado por ";".

codigoTipo;edad(entero);vip(1-TRUE;0-FALSE);familia(1-TRUE;0-FALSE);dia(entero);mes(entero)

Los código de tipo de entrada (que es igual al código de descuento) implementados para esta clase generadora son los siguientes:

0 – "niño"; 1 – "senior"; 2 – "carnet joven"; 3 – "discapacitados"; 4 – "estudiante"; 5 – "veterano"

La clase GeneradorVisitantes se encarga de decodificar el contenido del archivo Visitantes.txt y los importa dentro del objeto ParqueManager. En cada Tarea donde haga falta está incluido un archivo Visitantes.txt generado aleatoriamente. Todas las fechas están en el año 2019, para facilitar la generación de estadísticas en las Tareas 3 y 4.

Mas detalles se pueden encontrar en el Anexo II al final de la sección de la Tarea 4, junto con la clase GeneradorContenido

Archivos parque

Como se requiere en el enunciado, las Tareas 2, 3 y 4 cuentan con una clase parque con el método main. Esto sirve para demonstrar cierta funcionalidad básica del parque, a través de la creación de atracciones, importación de visitantes a través de *GeneradorVisitantes*, o la generación en pantalla de las estadísticas de las Tareas 3 y 4.

La implementación puede encontrarse en la sección de cada tarea en el Anexo II.

Anexo I: Detalles de las Clases y Notas de Implementación

Tarea 2

BuscadorDescuentos

Esta clase se encarga de almacenar los descuentos aplicables a las entradas del parque. Hace esto con un HashMap usando como clave de cada entrada el nombre del descuento (un String) y como valor el porcentaje del descuento. La clase también ofrece un método para buscar un descuento a partir de su clave identificadora, devolviendo el valor del descuento. Esto promueve un diseño dirigido por responsabilidad, y hace posible que cada objeto entrada pueda acceder a los descuentos apropiados, y que los descuentos se puedan modificar fácilmente, disminuyendo el acoplamiento y facilitando el mantenimiento.

Métodos públicos:

`public float getDescuento(String key)`
Devuelve un valor tipo float con el descuento buscado. Si el descuento no existe, se emite un mensaje de error, y se devuelve un descuento de valor 0, que no tendrá ningún efecto sobre el precio de la entrada.

`public void addDescuento(String key, float descuento)`
Método para añadir nuevos descuentos al objeto BuscadorDescuentos.

PARAMETROS

Como ya se ha descrito, esta clase contiene variables estáticas con algunos valores de uso general que pueden ser accedidos desde cualquier punto del programa. En la práctica, solo se incluyen como ejemplo de uso los descuentos generales aplicables a cualquier tipo de entrada, como por ejemplo el descuento para familias, o también el precio base de las entradas.

EntradaGen

Esta es la clase general para representar las entradas, y que implementa la interfaz EntradaIF. A parte de atributos básicos como el precio, el estado VIP y el estado Familia, contienen un HashMap para almacenar los descuentos aplicados, y un objeto Temporadas para buscar la temporada a la que pertenece.

Métodos públicos:

`public int getEdad()`
Devuelve la edad del visitante.

`public float getPrecio()`
Devuelve el precio de la entrada (el precio es modificado por los descuentos aplicados durante la creación de la entrada, ver clase MaquinaEntradas).

`public void applyDescuento(String tipo ,float descuento)`
Aplica descuentos al precio de la entrada. Sera llamado por la clase MaquinaEntradas. Se le debe pasar un el nombre del descuento y el valor del descuento (porcentaje aplicado al precio)

Nota de implementacion: Este metodo hace uso del HashMap descuentosApl de la clase EntradaGen para guardar los descuentos que han sido aplicados. Esto mantiene un record de los descuentos de cada entrada, para ser accedidos mas tarde.

```
public void applyDescuento(String tipo ,float descuento)
    {
        if (descuentoTotal < 90)
        {
            precio = precio * ((100 - descuento)/100);
            descuentosApl.put(tipo, descuento);
            descuentoTotal += descuento;
        }
        else
        {
            System.out.println("No se pueden aplicar mas descuentos");
        }
    }
```

`public String getTipo()`
Devuelve el tipo de entrada, que coincide con los descuentos aplicados. Así, una entrada con un descuento de senior y desempleado devolver una entrada de tipo senior – desempleado.

`public LocalDate getDate()`
Devuelve la fecha de la compra de la entrada en forma de un objeto LocalDate.

`public boolean getVIP()`
Indica si la entrada es una entrada VIP o no. Devuelve un boolean con el estado: TRUE si es VIP; FALSE si no es VIP

`public void setVIP()`
Convierte la entrada en una entrada VIP. Usa los porcentajes indicados en el enunciado para calcular el precio VIP para entradas senior y niños, además de para los demás tipos de entradas.

`public String getTemporada()`
Devuelve un string con la temporada en la que se ha comprado la entrada.

`public void setTemporada()`
Este método calcula a partir de la fecha de compra de la entrada y del objeto Temporadas suministrado por la MaquinaEntradas cuál es la temporada a la que pertenece la entrada. También aplica el descuento correspondiente al precio. Este método será llamado desde la MaquinaEntradas justo después de la creación de la entrada, modificando el precio inmediatamente antes de aplicar ningún descuento.

`public boolean getFamilia()`
Indica si la entrada es de tipo familiar o no. Devuelve un boolean: TRUE si la entrada es familiar; FALSE si la entrada no es familiar.

`public void setFamilia()`
Convierte la entrada en una entrada familiar, y aplica el descuento para entradas familiares (se encuentra en la clase PARAMETROS)

Niño

La clase niño implementa la interfaz y extiende la clase EntradaGen, heredando sus atributos y métodos y añadiendo la funcionalidad extra necesaria para implementar la entrada de un niño. La entrada Niño tiene todos los métodos de una entrada general, y además se le puede añadir referencia a la entrada de un acompañante que se especifica en el constructor de la clase.

Métodos públicos:

*Todos los métodos de la clase EntradaGen

`public EntradaGen getAcompañante()`
Devuelve la entrada del visitante acompañante del niño.

MaquinaEntradas

Esta clase se ocupa de la creación y configuración (VIP, descuentos etc.) de nuevas entradas. Esto se hace a través de dos métodos públicos principales.

Métodos públicos:

```
public EntradaIF nuevaEntrada(LocalDate fecha, int edad, boolean isVIP,
                              boolean isFamilia, String descuento)
```

Este método crea y devuelve una nueva entrada con la clase EntradaGen, y la configura según los datos suministrados.

Nota de implementacion: La creación y configuración se hace con el siguiente código. Se ha visto importante enseñar el código aquí para aclarar como se configuran las nuevas entradas. Como se ve, los descuentos aplicables se pasarían atraves de un String, cada descuento delimitado con una “,”.

```
EntradaIF entrada = new EntradaGen(fecha, edad, temporadas);
entrada.setTemporada();
entradaActual = entrada;

if (!descuento.equals("ninguno"))
{
    String tmp[] = descuento.split(",");
    for (int i = 0; i < tmp.length; i++)
    {
        setDescuento(tmp[i]);
    }
}


if (isVIP) {setVIP();}
if (isFamilia) {setFamilia();}

ultimoAdulto = (EntradaGen)entrada;
return entrada;
```

```
public EntradaIF nuevaEntradaNiño(LocalDate fecha, int edad, boolean
                                  isVIP, boolean isFamilia, String descuento)
```

Funciona de manera similar al método nuevaEntrada, solo que usa el campo *ultimoAdulto* del objeto *MaquinaEntrada* para designar un acompañante al niño. Si no existe un ultimoAdulto, se generará un mensaje de error.

AtraccionA

Esto es un ejemplo de implementación de la interfaz AtraccionIF. Se han implementado otros tipos de atracciones, pero la funcionalidad que ofrecen es la misma, y por lo tanto solo se describirán los métodos públicos de este ejemplo.

La clase *AtraccionA* suministra información sobre el tipo de atracción “A”, tal y como fue especificado en el enunciado. También almacena en dos estructuras List los visitantes que han visitado la atracción y los trabajadores que trabajan en ella.

Métodos públicos

```
public String getTipo()
```

Devuelve el tipo de atracción en formato String; ejemplo: “A”.

`public boolean getVIP()`
Indica si la atracción tiene acceso VIP. Devuelve TRUE si tiene acceso VIP; FALSE si no lo tiene

`public void setVIP(boolean VIP)`
Modifica el estado accesoVIP de la atracción. Esto permite modificar la plantilla de la clase atracción para crear nuevos subtipos de la atracción modificados, añadiendo más flexibilidad a la clase. El mismo diseño se usará en otros métodos.

`public float getMinAlturaCM()`
Devuelve altura mínima de acceso a la atracción. Si no hay altura mínima, devuelve 0.

`public void setMinAlturaCM(float min)`
Modifica la altura mínima de acceso de la atracción.

`public float getMaxAlturaCM()`
Devuelve la altura máxima de acceso a la atracción en CM. Si no hay altura máxima de acceso devuelve un 0.

`public void setMaxAlturaCM(float max)`
Modifica la altura máxima de acceso a la atracción.

`public int getEdad()`
Devuelve la edad mínima de acceso a la atracción. Si no hay edad mínima devuelve un 0.

`public void setMinEdad(int min)`
Modifica la edad mínima de acceso a la atracción.

`public boolean getAccesoNiños()`
Indica si se permite el acceso a visitantes con entrada Niño a la atracción. Devuelve TRUE si se permite; FALSE si no se permite.

`public void setAccesoNiños(boolean acceso)`
Modifica el acceso a niños a la atracción.

`public boolean getAccesoAdultos()`

Indica si la atracción permite el acceso a visitantes que no sean niños. Devuelve TRUE si se permite; FALSE si no se permite.

`public void setAccesoAdultos(boolean acceso)`
Modifica el acceso a no-niños a la atracción.

`public void setNumRespAtracc(int num)`
Modifica el número de Responsables de Atraccion necesarios para el uso de la atracción.

`public int getNumRespAtracc()`
Devuelve el número de Responsables de Atraccion necesarios para el uso de la atracción.

`public void setNum AyuAtracc(int num)`
Modifica el número de Ayudantes de Atraccion necesarios para el uso de la atracción.

`public int getNum AyuAtracc()`
Devuelve el número de Ayudantes de Atraccion necesarios para el uso de la atracción.

`public void addTrabajador(Trabajador trabajador)`
Añade un nuevo trabajador a la lista de trabajadores de la atracción.

`public List getTrabajadores()`
Devuelve la lista de trabajadores de la atracción.

`public void usar(EntradaIF entrada)`
Añade un nuevo visitante a la lista de visitantes de la atracción. También se encarga de verificar que el visitante tenga acceso a la atracción a partir de su tipo de entrada. Si el acceso es denegado se imprimirá un mensaje de error.

`public List getUsuarios()`
Devuelve la lista de visitantes que han visitado la atracción.

CreadorAtracciones

Esta clase se encarga de crear nuevas atracciones. Consta de un solo método público, cuya implementación se analiza a continuación.

Métodos públicos:

`public AtraccionIF nuevaAtraccion(String tipo)`
Este método crea y devuelve una nueva atracción del tipo especificado en el parámetro tipo.

Nota de implementacion: Este metodo usa una estructura *switch* para seleccionar que tipo de atracción debe ser creado:

```
switch(tipo) {
            case "A":
                atraccionActual = new AtraccionA();
                break;
            case "B":
                atraccionActual = new AtraccionB();
                break;
            case "C":
                atraccionActual = new AtraccionC();
                break;
            case "D":
                atraccionActual = new AtraccionD();
                break;
            case "E":
                atraccionActual = new AtraccionE();
                break;
            default:
                atraccionActual = new AtraccionA();
        }
```

Extensiones de la Clase Abstracta Trabajador

Ejemplo AyudanteAtraccion

Las clases AyudanteAtraccion, RespAtraccion, RelPublicas y AtencionCl extienden a la clase abstracta Trabajador, implementando dos métodos públicos: getTipo() y getSueldo(). La implementación de los dos métodos es idéntica, solo cambian los atributos tipo y sueldo dependiendo del tipo de trabajador.

Es de notar que para el tipo de trabajador se usa la clase enum TiposTrabajadores.

Métodos públicos:

`public TiposTrabajadores getTipo()`
Devuelve el tipo de trabajador usando la clase enum TiposTrabajadores.

`public float getSueldo()`
Devuelve el sueldo del trabajador.

CreadorTrabajadores

De forma similar a la clase *CreadorAtracciones*, esta clase se encarga de la creación de nuevos trabajadores, sirviéndose de una estructura *switch* para seleccionar el tipo de trabajador deseado.

Es de notar que para el tipo de trabajador se usa la clase enum TiposTrabajadores.

Métodos públicos:

`public Trabajador nuevoTrabajador(TiposTrabajadores tipo)`
Crea y devuelve un nuevo trabajador del tipo especificado. Para el tipo se usa la clase enum TiposTrabajadores.

ParqueManager

Esta clase se ocupa de almacenar y gestionar las estructuras y funcionalidades principales del parque. Sería la clase que en futuro se pudiese conectar a una interfaz de texto o interfaz gráfica, ya que provee la funcionalidad principal del parque.

Métodos públicos:

```
public void addEntrada(LocalDate fecha, int edad, boolean isVIP,
                       boolean isFamilia, String descuento)
```

Crea una nueva entrada (EntradaGen), la configura con los parámetros, y la añade a la lista de entradas. Los descuentos, como ya se ha mencionado, se especifican con una cadena String, cada descuento delimitado por una coma “,” sin espacios.

```
public void addEntradaNiño(LocalDate fecha, int edad, boolean isVIP,
                           boolean isFamilia, String descuento)
```

Crea una nueva entrada niño (Niño) la configura con los parámetros, y la añade a la lista de entradas. Los descuentos, como ya se ha mencionado, se especifican con una cadena String, cada descuento delimitado por una coma “,” sin espacios.

```
public void addTrabajador(TiposTrabajadores tipo)
```

Crea un nuevo trabajador y lo añade a la lista de trabajadores.

```
public void addAtraccion(String tipo)
```

Crea una nueva atracción y la añade a la lista de atracciones. También se encarga de generar los trabajadores necesarios para el funcionamiento de la atracción, y los añade a la lista de trabajadores. Los trabajadores también se añaden a las listas de trabajadores internas de las atracciones

Nota de implementacion: Se hace nota de la implementacion interna de este metodo para entender como se generan los trabajadores de la atraccion

```
AtraccionIF atraccion = generadorAtracciones.nuevaAtraccion(tipo);
        Trabajador nuevoTrabajador;
        for (int i = 0; i < atraccion.getNumRespAtracc(); i++)
        {
            nuevoTrabajador =
generadorTrabajadores.nuevoTrabajador(TiposTrabajadores.RESP_ATRACC);
            TrabajadoresParque.add(nuevoTrabajador);
            atraccion.addTrabajador(nuevoTrabajador);
        }

        for (int i = 0; i < atraccion.getNumAyuAtracc(); i++)
        {
            nuevoTrabajador =
generadorTrabajadores.nuevoTrabajador(TiposTrabajadores.AYU_ATRACC);
            TrabajadoresParque.add(nuevoTrabajador);
            atraccion.addTrabajador(nuevoTrabajador);
        }

        AtraccionesParque.add(atraccion);
```

```
public int getNumTrabajadores(TiposTrabajadores tipo)
```

Devuelve el número de trabajadores de un cierto tipo contratados en el parque.

```
public int getNumTrabajadores()
```

Devuelve el total de trabajadores del parque.

`public int getNumVisitantes()`
Devuelve el total de visitantes registrados en el parque.

`public void usarAtraccion(int index, EntradaIF entrada)`
Método de apoyo para usar las atracciones. Esto se usará cuando se generen usos de atracciones aleatorios para probar el sistema.

`public void randomUsarAtracciones()`
Método de apoyo que genera un uso aleatorio de las atracciones del parque. Esto se usará sobre todo para probar las estadísticas de las tareas posteriores.

`public void setContenidoAtraccionesFuncionando()`
Método de apoyo que se ocupa de generar contenido para el objeto de clase AtraccionesFuncionando. Esto se usará en la Tarea 4 para generas estadísticas sobre los gastos de personal diario. El método divide las atracciones en dos periodos de funcionamiento (las dos mitades del año 2019).

`public AnalizadorEstadisticas analisisEstadistico()`
Devuelve un objeto AnalizadorEstadisticas. Esto se usará en las tareas 3 y 4 para generar estadísticas sobre diversos aspectos del parque

`public void trabajadoresPorAtraccion()`
Imprime informacion sobre los trabajadores por atraccion del parque.

`public void resumenAtraccionesActivas()`
Imprime informacion sobre los periodos de funcionamiento de las atracciones del parque

`public HashMap<String, Float> obtenerDescuentos()`
Devuelve la lista de descuentos aplicables en el parque, de la clase ObtenerDescuentos

AnalizadorEstadisticas – Tarea 3

Esta clase recibe del objeto *ParqueManager* los datos en forma de lista de las estructuras del parque, los trabajadores, y los visitantes del parque (lista de entradas registradas). Con estos datos genera varios resúmenes estadísticos, imprimiendo la información en pantalla.

Métodos públicos:

`public void resumenVisitantes(int year)`
Genera un resumen del número de visitantes del parque, agrupado por día, semana, mes y año, del año especificado (year). Imprime el resumen en pantalla

Notas de implementación: Este método, como otros en la clase *AnalizadorEstadisticas*, usa un bucle *for* junto con condicionales anidados para agrupar los datos por dia, semana, mes y año. Empieza con la primer fecha de la lista de visitantes:

```
int dia = ListaEntradas.get(0).getDate().getDayOfMonth();
```

…
Y continua con un bucle *for-each* recorriendo cada entrada de la lista

```
for (EntradaIF entrada : ListaEntradas)
```

Por cada entrada, verifica si coincide con la fecha de las entradas precedentes:

```
            if (fecha.getYear() == anno)
            {
                if (fecha.getMonthValue() == month) //Loop para el mes
                {
                    if (fecha.get(woy) == semana)
                    {
                         if (fecha.getDayOfMonth() == dia)
                         {
```

Y usa variable contadores para contar los visitantes de cada grupo. Cuando se detecta que la entrada actual pertenece a un nuevo periodo (nuevo dia, o semana etc…) se imprime la información de los contadores apropiados y estos se ponen a cero para seguir contando un nuevo grupo.

El código se ha diseñado de manera que se pueda reutilizar en la mayor parte posible para los otros métodos que necesitan agrupar los resultados por dia, semana, mes o año.

También se han usado funciones de soporte para facilitar ciertos cálculos repetitivos, como por ejemplo los promedios. Esto se hace a través de métodos privados como

```
private float promedioMensual(int n, int m)
```

`public void resumenPrecios(int year)`
Genera un resumen de los promedios diarios, semanales, mensuales y anuales de los precios de las entradas del año especificado (year). La estructura es muy similar al del método *resumenVisitantes()*. Los resultados se imprimen en pantalla. Para más detalles de implementación consultar el código fuente consulte el Anexo II.

`public void resumenVisitasAtracciones(int year, AtraccionIF atraccion)`
Genera un resumen de las visitas a la atraccion especificada en el año especificado. Se imprime un resumen de las visitas agrupadas por día, semana, mes y año.

Para más detalle sobre el código fuente consulte el Anexo II.

`public void resumenAtracciones()`
Este método simple imprime en pantalla un recuento de las atracciones del parque, agrupadas por tipo de atracción.

Notas de implementación: para ser lo mas flexible posible, y acomodar nuevos tipos de atracción en el futuro, el recuento se hace con un HashMap. La clave es el tipo de atracción (una cadena String), y el valor el numero de atracciones existentes de ese tipo. Cada vez que se encuentra un tipo nuevo se crea una nueva entrada en el HashMap. Para mas detalles consultar el código fuente en el Anexo II. Este sistema se usa para añadir flexibilidad y reducir acoplamiento en otros métodos, como en *resumenVisitantesTipo()*

`public int resumenTrabajadoresTipo(TiposTrabajadores tipo)`
Este método devuelve un entero con el número de trabajadores registrados en el parque pertenecientes a la categoría (tipo) indicada.

`public void resumenVisitantesTipo()`
Este método imprime en pantalla un resumen del número de visitantes registrados del parque, organizados por tipo de entrada comprada.

AtraccionesFuncionando – Tarea 4

Esta clase se ocupa de almacenar y suministrar información sobre las atracciones activas en diferentes periodos del año. Su existencia es requerida por el enunciado (Tarea 4) y promueve un uso de diseño dirigido por responsabilidad. Los datos están almacenados en un HashMap, con los periodos de tiempo (*PeriodoTemporada*) como clave y la lista de atracciones activas como valor asociado.

Métodos públicos:

```
public List<AtraccionIF> getAtracciones(LocalDate fecha)
```

Devuelve una List<AtraccionIF> con todas las atracciones activas en la fecha especificada.

```
public void addAtraccion(PeriodoTemporada periodo, List<AtraccionIF>
atracciones)
```

Método para añadir una lista de atracciones activas asociado un periodo del año especifico. El método verifica si dicho periodo exista ya o no. Si existe, actualiza la lista de atracciones activas, añadiendo las nuevas atracciones. Si el periodo no existe, se añade en el HashMap de la clase. Para que coincidan dos periodos, deben coincidir la fecha inicial y la fecha final. No puede haber sobreposiciones. Si se detectan sobreposiciones, la nueva entrada será rechazada con un mensaje de error.

AnalizadorEstadisticas (Expansión Tarea 4)

En la tarea 4, esta clase se expande para añadir un nuevo método público:

public void resumenGastoPersonal(int year).

Para suministrar los datos que emplea este método, también se añade un nuevo campo al constructor: AtraccionesFuncionando atraccionesAct, que es un objeto de clase AtraccionesFuncionando que contendrá las atracciones activas para especificadas para diferentes periodos del año.

Métodos públicos (expansión);

```
public void resumenGastoPersonal(int year)
```

Genera un resumen de los gastos de personal del parque para un año especifico, subdividiéndolos por semana, mes y año, y calculando promedios para cada categoría. Los resultados se imprimen en pantalla a través de la consola.

MenuInterface (Programa Completo Tarea 4)

Clase con todos los menus del programa, que actúa de interfaz entre el objeto ParqueManager y el usuario. Tambien se encarga de verificar la mayoría de los datos de entrada antes de que estos sean usados por la funcionalidad del programa.

Metodos Publicos:

`public void menu1()`
Menu principal del programa. Desde aquí se pueden acceder a todos los menús y por tanto a toda la funcionalidad del programa completo.

Anexo II: Código fuente de las clases

Tarea 1:

EntradaIF

```
/**
 * Interfaz principal de las entradas, con
 * toda la funcionalidad necesaria.
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */
import java.time.*;

public interface EntradaIF
{
    /**
     * Method getPrecio
     *
     * Metodo que devuelve el precio
     * de la entrada
     *
     * @return Precio de la entrada
     */
    float getPrecio();

    /**
      * Method getDate
      *
      * Devuelve la fecha de compra
      * de la entrada
      *
      * @return Fecha de la compra (LocalDate)
      */
    LocalDate getDate();

    /**
     * Method getVIP
     *
     * Indica si la entrada es una
     * entrada VIP o no
     *
     * @return True si la entrada es VIP
     */
    boolean getVIP();

    /**
```

```
 * Method setVIP
 *
 * Convierte entrada en entrada VIP
 * Utiliza porcentajes especificados en el
 * enunciado para calcular el precio de
 * la entrada VIP para niños y seniors
 *
 */
void setVIP();

/**
 * Method getTemporada
 *
 * Devuelve la temporada en la que se ha
 * comprado la entrada
 *
 * @return Temporada en la que se ha comprado la entrada (String)
 */
String getTemporada();

/**
 * Method setTemporada
 *
 * Metodo para que la entrada calcule
 * la temporada en la que se compra
 * a partir de la lista de temporadas suministrada
 * por la MaquinaEntradas
 *
 */
void setTemporada();

 /**
 * Method getFamilia
 *
 * Indica si la entrada es una entrada
 * familiar o no
 *
 * @return True: Si la entrada es familiar
 */
boolean getFamilia();

/**
 * Method setFamilia
 *
 * Convierte la entrada en una entrada familiar,
 * aplicando el descuento apropiado
 *
 */
void setFamilia();
```

```
    /**
     * Method applyDescuento
     *
     * Metodo para aplicar descuentos a la
     * entrada, modificando el precio.
     *
     * @param tipo Identificador del descuento en formato string
     * @param descuento Valor del descuento (porcentaje)
     */
    void applyDescuento(String tipo, float descuento);

    /**
     * Method getTipo
     *
     * Devuelve un objeto string con el
     * tipo de entrada. Esto se construye a partir
     * de los descuentos aplicados, que identifican
     * la entrada
     *
     * @return El tipo de entrada
     */
    String getTipo();

    /**
     * Metodo getter que devuelve edad del visitante
     *
     * @return    edad del visitante
     */
    int getEdad();

}
```

AtraccionIF

```
/**
 * Write a description of class Atraccion here.
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */

import java.util.List;
import java.util.LinkedList;
public interface AtraccionIF
{


    /**
     * Method getTipo
     *
     * Devuelve el tipo de atraccion
     *
     * @return Tipo de atraccion (String)
     */
    public String getTipo();

    /**
     * Method getVIP
     *
     * Indica si la atraccion tiene
     * acceso VIP.
     *
     * @return True si la atraccion tiene acceso VIP
     */
    public boolean getVIP();

    /**
     * Method setVIP
     *
     * Modifica el atributo accesoVIP
     * de la atraccion
     *
     * @param VIP nuevo estado accesoVIP (boolean)
     */
    public void setVIP(boolean VIP);

    /**
     * Method getMinAlturaCM
     *
     * Devuelve la altura minima
     * de acceso a la atraccion
     * en CM
```

```
 * Si no hay altura minima devuelve 0.
 *
 * @return Altura minima para acceder a la atraccion
 */
public float getMinAlturaCM();

/**
 * Method setMinAlturaCM
 *
 * Modifica la altura minima
 * de acceso a la atraccion
 *
 * @param min Nueva altura minima
 */
public void setMinAlturaCM(float max);

/**
 * Method getMaxAlturaCM
 *
 * Devuelve la altura maxima de acceso a la
 * atraccion en CM
 * Si no hay altura maxima devuelve 0
 *
 * @return Altura maxima en CM
 */
public float getMaxAlturaCM();

/**
 * Method setMaxAlturaCM
 *
 * Modifica la altura maxima de acceso
 * a la atraccion
 *
 * @param max Altura maxima de acceso a la atraccion
 */
public void setMaxAlturaCM(float max);

/**
 * Method getEdad
 *
 * Devuelve la edad minima de acceso a la atraccion
 * Si no hay edad minima devuelve 0
 *
 * @return Edad minima de acceso a la atraccion
 */
public int getEdad();

/**
 * Method setMinEdad
 *
```

```
 * Modifica la edad minima
 * de entrada a la atraccion
 *
 * @param min Edad minima de entrada a la atraccion
 */
public void setMinEdad(int min);

/**
 * Method getAccesoNiños
 *
 * Indica si la atraccion permite
 * el acceso a visitantes con entrada
 * de niño
 *
 * @return Acceso de niños - True si se permite el acceso
 */
public boolean getAccesoNiños();

/**
 * Method setAccesoAdultos
 *
 * Modifica el acceso a no-niños
 * a la atraccion
 *
 * @param acceso Acceso a no-niños (boolean)
 */
public void setAccesoNiños(boolean acceso);

/**
 * Method getAccesoAdultos
 *
 * Indica si la atraccion permite
 * el acceso de visitantes que no
 * sean niños
 *
 * @return Acceso de adultos - True si se permite
 */
public boolean getAccesoAdultos();

/**
 * Method setAccesoNiños
 *
 * Modifica el acceso a niños
 * de la atraccion
 *
 * @param acceso Acceso a niños (boolean)
 */
public void setAccesoAdultos(boolean acceso);

/**
```

```
 * Method setNumRespAtracc
 *
 * Modifica el numero de Responsables
 * de Atraccion necesarios para el uso de la
 * atraccion
 *
 * @param num Numero de Responsables de Atraccion necesarios
 */
public void setNumRespAtracc(int num);

 /**
 * Method getNumRespAtracc
 *
 * Devuelve el numero de Responsables
 * de Atraccion necesarios para el uso de la
 * atraccion.
 *
 * @return Numero de Responsables de Atraccion necesarios
 */
public int getNumRespAtracc();

/**
 * Method setNumAyuAtracc
 *
 * Modifica el numero de Ayudantes de
 * Atraccion necesario para el uso de la
 * atraccion
 *
 * @param num Numero de Ayudantes de Atraccion necesarios
 */
public void setNumAyuAtracc(int num);

/**
 * Method getNumAyuAtracc
 *
 * Devuelve el numero de Ayudantes de
 * Atraccion necesario para el uso de la
 * atraccion
 *
 * @return Numero de Ayudantes de Atraccion necesarios
 */
public int getNumAyuAtracc();

/**
 * Method addTrabajador
 *
 *  Añade un nuevo trabajador a la lista
 *  de trabajadores de la atraccion
 *
 * @param trabajador Nuevo trabajador
```

```
     */
    public void addTrabajador(Trabajador trabajador);

     /**
     * Method getTrabajadores
     *
     * Devuelve la lista de trabajadores de la atraccion
     *
     * @return Lista de trabajadores de la atraccion
     */
    public List getTrabajadores();

    /**
     * Method usar
     *
     * Añade un nuevo visitante a la lista de
     * visitantes de la atraccion.
     * Tambien se encarga de verificar que el visitante
     * tenga acceso a la atraccion a partir de su entrada
     *
     * @param entrada Entrada del visitante
     */
    public void usar(EntradaIF entrada);

    /**
     * Method getUsuarios
     *
     *  Devuelve la lista de visitantes
     *  de la atraccion
     *
     * @return Lista de visitantes de la atraccion
     */
    public List getUsuarios();

}
```

Trabajador (Clase abstracta)

```
/**
 * Clase abstracta Trabajador con metodos
 * comunes a todos los trabajadores.
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */

import java.util.List;
import java.util.LinkedList;

public abstract class Trabajador
{
   private List<AtraccionIF> atracciones;

   public Trabajador()
   {
       atracciones = new LinkedList<AtraccionIF>();
   }


   /**
    * Method getTipo
    *
    * Devuelve el tipo de trabajador
    *
    * @return Tipo de trabajador
    */
   abstract public TiposTrabajadores getTipo();

   public List<AtraccionIF> getAtracciones()
   {
       return atracciones;
   }

   /**
    * Method setAtracciones
    *
    * Añade una lista de atracciones
    * al atributo atracciones del trabajador
    *
    * @param atracciones Lista de atracciones
    */
   public void setAtracciones(LinkedList<AtraccionIF> atracciones)
   {
       this.atracciones = new LinkedList(atracciones);
   }
```

```
/**
 * Method addAtraccion
 *
 * Añade una nueva atraccion a la lista
 * de atracciones
 *
 * @param atraccion A parameter
 */
public void addAtraccion(AtraccionIF atraccion)
{
    atracciones.add(atraccion);
}

/**
 * Method removeAtraccion
 *
 * Elimina una atraccion especifica de la lista
 * de atracciones
 *
 * @param atraccion Atraccion que se debe eliminar
 */
public void removeAtraccion(AtraccionIF atraccion)
{
    if (atracciones.contains(atraccion))
    {
        atracciones.remove(atraccion);
    }

}

/**
 * Method clearAtracciones
 *
 * Elimina todas las atracciones de la lista
 * de atracciones
 *
 */
public void clearAtracciones()
{
    atracciones.clear();
}

/**
 * Method numeroDeAtracciones
 *
 * Devuelve el numero de atracciones
 * en la lista de atracciones del
 * trabajador
 *
 * @return Numero de atracciones
```

```
     */
    public int numeroDeAtracciones()
    {
        return atracciones.size();
    }

    /**
     * Method getSueldo
     *
     * Devuelve el sueldo del trabajador
     *
     * @return Sueldo del trabajador
     */
    abstract public float getSueldo();
}
```

Tarea 2:

BuscadorDescuentos

```
/**
 * Clase que se ocupa de almacenar y suministrar los posibles
 * descuentos de las entradas.
 *
 * @author (Samue Alarco)
 * @version (v1.0)
 */
import java.util.HashMap;

public class BuscadorDescuentos
{
    private HashMap<String, Float> mapaDescuentos ;

    /**
     * Constructor for objects of class BuscadorDescuentos
     */
    public BuscadorDescuentos()
    {
        // initialise instance variables
        mapaDescuentos = new HashMap<String, Float>();
        mapaDescuentos.put("carnet joven", 10f);
        mapaDescuentos.put("discapacitado", 20f);
        mapaDescuentos.put("estudiante", 10f);
        mapaDescuentos.put("veterano", 10f);
        mapaDescuentos.put("senior", 35f);
    }

    /**
     * Method getDescuento
     *
     * Metodo para buscar un descuento especifico
     *
     * @param key Clave del descuento
     * @return Valor (porcentaje) del descuento
     */
    public float getDescuento(String key)
    {
        if (mapaDescuentos.containsKey(key))
        {
            return mapaDescuentos.get(key);
        }
        else
        {
            System.out.println("Descuento no existe");
            return 0f; // No se aplica ningun descuento
        }
```

```
    }

    /**
     * Method addDescuento
     *
     * Metodo para añdir nuevos descuentos
     * al objeto
     *
     * @param key Identificacion del descuento (String)
     * @param descuento Valor del descuento (porcentaje a descontar)
     */
    public void addDescuento(String key, float descuento)
    {
        if (descuento < 1)
        {
            System.out.println("Descuento no valido");
        }
        else
        {
            mapaDescuentos.put(key, descuento);
        }
    }
}
```

PARAMETROS

```
/**
 * Clase que se ocupa de concretizar ciertos parametros de uso
 * general, especificamente los que tengan que ver con los precios
 * de las entradas  varios descuentos.
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */
public class PARAMETROS
{
    public static float PRECIO_BASE = 60f;
    public static float DESCUENTO_NIÑO = 50f;
    public static float PRECIO_VIP_BASE = 50f;
    public static float DESCUENTO_FAMILIA = 10f;
}
```

EntradaGen

```
/**
 * Implementacion general de la interfaz EntradaIF.
 * Ofrece la funcionalidad general de las entradas
 * del parque de atracciones
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */

import java.util.HashMap;
import java.time.*;

public class EntradaGen implements EntradaIF
{
    // instance variables - replace the example below with your own
    protected HashMap<String, Float> descuentosApl;
    protected float precio;
    protected float descuentoTotal;

    private LocalDate fecha;
    private int edad;

    private boolean isVIP;
    private boolean VIPAplicado = false;
    private boolean isFamilia;
    private String temporada;

    private Temporadas temporadas;

    /**
     * Constructor for objects of class EntradaGen
     */
    public EntradaGen(LocalDate fecha, int edad, Temporadas temporadas)
    {
        this.fecha = fecha;
        this.edad = edad;
        precio = PARAMETROS.PRECIO_BASE;
        descuentosApl = new HashMap<String, Float>();
        this.temporadas = temporadas;
    }

    /**
     * Metodo getter que devuelve edad del visitante
     *
     * @return    edad del visitante
     */
    public int getEdad()
```

```
{
    // return edad
    return this.edad;
}

/**
 * Method getPrecio
 *
 * Metodo que devuelve el precio
 * de la entrada
 *
 * @return Precio de la entrada
 */
public float getPrecio()
{
    return precio;
}

/**
 * Method applyDescuento
 *
 * Metodo para aplicar descuentos a la
 * entrada, modificando el precio.
 *
 * @param tipo Identificador del descuento en formato string
 * @param descuento Valor del descuento (porcentaje)
 */
public void applyDescuento(String tipo ,float descuento)
{
    if (descuentoTotal < 90)
    {
        precio = precio * ((100 - descuento)/100);
        descuentosApl.put(tipo, descuento);
        descuentoTotal += descuento;
    }
    else
    {
        System.out.println("No se pueden aplicar mas descuentos");
    }
}

/**
 * Method getTipo
 *
 * Devuelve un objeto string con el
 * tipo de entrada. Esto se construye a partir
 * de los descuentos aplicados, que identifican
 * la entrada
 *
 * @return El tipo de entrada
```

```
 */
public String getTipo()
{
    String tipos = "";
    for (String key : descuentosApl.keySet())
    {
        tipos += key + " - ";
    }
    tipos = tipos.substring(0, tipos.length()-3);
    return tipos;
}


 /**
  * Method getDate
  *
  * Devuelve la fecha de compra
  * de la entrada
  *
  * @return Fecha de la compra (LocalDate)
  */
 public LocalDate getDate()
{
    return this.fecha;
}

/**
 * Method getVIP
 *
 * Indica si la entrada es una
 * entrada VIP o no
 *
 * @return True si la entrada es VIP
 */
public boolean getVIP()
{
    return this.isVIP;
}

/**
 * Method setVIP
 *
 * Convierte entrada en entrada VIP
 * Utiliza porcentajes especificados en el
 * enunciado para calcular el precio de
 * la entrada VIP para niños y seniors
 *
 */
public void setVIP()
{
```

```
    isVIP = true;
    if (!VIPAplicado)
        {
            if (descuentosApl.containsKey("senior"))
            {
                precio += (precio * (0.83333));
            } else if (descuentosApl.containsKey("niño"))
            {
                System.out.println("descuento niño");
                precio += (precio * (0.83333));
            } else
            {
                precio += PARAMETROS.PRECIO_VIP_BASE;
            }

            VIPAplicado = true;
        }
}

/**
 * Method getTemporada
 *
 * Devuelve la temporada en la que se ha
 * comprado la entrada
 *
 * @return Temporada en la que se ha comprado la entrada (String)
 */
public String getTemporada()
{
    return this.temporada;
}

/**
 * Method setTemporada
 *
 * Metodo para que la entrada calcule
 * la temporada en la que se compra
 * a partir de la lista de temporadas suministrada
 * por la MaquinaEntradas
 *
 */
public void setTemporada()
{
    this.temporada = temporadas.checkTemporada(fecha);
    precio = precio * (temporadas.get(temporada) / 100);
}

/**
 * Method getFamilia
 *
```

```
     * Indica si la entrada es una entrada
     * familiar o no
     *
     * @return True: Si la entrada es familiar
     */
    public boolean getFamilia()
    {
        return this.isFamilia;
    }

    /**
     * Method setFamilia
     *
     * Convierte la entrada en una entrada familiar,
     * aplicando el descuento apropiado
     *
     */
    public void setFamilia()
    {
        isFamilia = true;
        if (isFamilia)
            {
                applyDescuento("Familia" ,PARAMETROS.DESCUENTO_FAMILIA);
            }
    }
}
```

Niño

```
/**
 * Clase para las entradas tipo Niño.
 * Extiende a la clase EntradaGen e
 * implementa la interfaz EntradaIF
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */
import java.util.HashMap;
import java.time.*;


public class Niño extends EntradaGen implements EntradaIF
{
    private EntradaGen acompañante;

    /**
     * Constructor for objects of class Niño
     */
    public Niño(LocalDate fecha ,int edad, Temporadas
temporadas,EntradaGen acompañante)
    {
        super(fecha, edad, temporadas);
        this.acompañante = acompañante;
        super.applyDescuento("niño", PARAMETROS.DESCUENTO_NIÑO);
    }


    /**
     * Method getAcompañante
     *
     * Devuelve la entrada del acompañante
     * del niño
     *
     * @return Entrada del acompañante del niño..
     */
    public EntradaGen getAcompañante()
    {
        return acompañante;
    }
}
```

MaquinaEntradas

```
/**
 * Clase MaquinaEntradas: se encarga de gestionar la asignacion
 * de entradas a cada visitante, gestionando la aplicacion de
 * descuentos y la seleccion de la entrada apropiada para cada
 * visitante
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */

import java.time.LocalDate;

public class MaquinaEntradas
{
   private EntradaIF entradaActual;
   private EntradaGen ultimoAdulto;
   private BuscadorDescuentos buscadorDescuentos;
   private Temporadas temporadas;


    /**
     * Constructor for objects of class MaquinaEntradas
     */
    public MaquinaEntradas()
    {
        buscadorDescuentos = new BuscadorDescuentos();
        temporadas = new Temporadas();
    }


    /**
     * Method nuevaEntrada
     *
     * Crea y devuelve una nueva entrada general
     * EntradaGen
     *
     * @param fecha Fecha de compra
     * @param edad Edad de la persona
     * @param isVIP Si es entrada VIP
     * @param isFamilia Si es entrada Familiar
     * @param descuento Lista de descuentos aplicables
     * @return Nueva entrada (EntradaIF)
     */
    public EntradaIF nuevaEntrada(LocalDate fecha, int edad,
                                  boolean isVIP,
                                 boolean isFamilia, String descuento)
    {
```

```
    EntradaIF entrada = new EntradaGen(fecha, edad, temporadas);
    entrada.setTemporada();
    entradaActual = entrada;
    if (isFamilia) {setFamilia();}
    if (!descuento.equals("ninguno"))
    {
        String tmp[] = descuento.split(",");
        for (int i = 0; i < tmp.length; i++)
        {
            setDescuento(tmp[i]);
        }

    }

    if (isVIP) {setVIP();}

    ultimoAdulto = (EntradaGen)entrada;
    return entrada;
}

private void setFamilia()
{
    if (entradaActual != null)
    {
        entradaActual.setFamilia();
    }
}

private void setVIP()
{
    if (entradaActual != null)
    {
        entradaActual.setVIP();
    }
}

private void setDescuento(String descuento)
{
    float fDescuento =  buscadorDescuentos.getDescuento(descuento);
    entradaActual.applyDescuento(descuento, fDescuento);
}


/**
 * Method nuevaEntradaNiño
 *
 * Metodo para crear una nueva entrada
 * de niño. Usa el campo ultimo adulto
 * para añadir a la entrada un acompañante
 *
```

```
     * @param fecha Fecha de compra
     * @param edad Edad de la persona
     * @param isVIP Si es entrada VIP
     * @param isFamilia Si es entrada Familiar
     * @param descuento Lista de descuentos aplicables
     * @return Nueva entrada (EntradaIF)
     */
    public EntradaIF nuevaEntradaNiño(LocalDate fecha, int edad,
                                boolean isVIP,
                               boolean isFamilia, String descuento)
    {
        if(ultimoAdulto != null)
        {
          EntradaIF entrada = new Niño(fecha, edad, temporadas,
ultimoAdulto);
          entrada.setTemporada();
          if (isFamilia) {setFamilia();}

          if (!descuento.equals("ninguno")) {setDescuento(descuento);}
          if (isVIP) {setVIP();}
          return entrada;
        }
        else
        {
          System.out.println("Se necesita un adulto");
          return null;
        }

    }

}
```

Ejemplo de Clase de Atraccion (AtraccionA)

```
/**
 * Implementacion de la Atraccion Tipo A.
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */

import java.util.List;
import java.util.LinkedList;
public class AtraccionA implements AtraccionIF
{
    private String tipo;
    private boolean accesoVIP;
    private float minAlturaCM, maxAlturaCM;
    private int minEdad;
    private boolean accesoNiños;
    private boolean accesoAdultos;
    private int numRespAtracc;
    private int numAyuAtracc;
    private int contador;

    private List trabajadores;
    private List usuarios;


    /**
     * Constructor for objects of class AtraccionTipo
     */
    public AtraccionA()
    {
        this.tipo = "A";
        trabajadores =  new LinkedList<Trabajador>();
        usuarios =  new LinkedList<EntradaIF>();
        // Valores default
        accesoVIP = true;
        minAlturaCM = 120; // no minimo de altura
        maxAlturaCM = 0;
        minEdad = 0; // no minimo de edad;
        accesoNiños = true;
        accesoAdultos = true;
        numRespAtracc = 1;
        numAyuAtracc = 6;
    }

    /**
     * Method getTipo
     *
```

```
 * Devuelve el tipo de atraccion
 *
 * @return Tipo de atraccion (String)
 */
public String getTipo()
{
  return this.tipo;
}

/**
 * Method getVIP
 *
 * Indica si la atraccion tiene
 * acceso VIP.
 *
 * @return True si la atraccion tiene acceso VIP
 */
public boolean getVIP()
{
    return accesoVIP;
}

/**
 * Method setVIP
 *
 * Modifica el atributo accesoVIP
 * de la atraccion
 *
 * @param VIP nuevo estado accesoVIP (boolean)
 */
public void setVIP(boolean VIP)
{
    this.accesoVIP = VIP;
}

/**
 * Method getMinAlturaCM
 *
 * Devuelve la altura minima
 * de acceso a la atraccion
 * en CM
 * Si no hay altura minima devuelve 0.
 *
 * @return Altura minima para acceder a la atraccion
 */
public float getMinAlturaCM()
{
    return this.minAlturaCM;
}
```

```
/**
 * Method setMinAlturaCM
 *
 * Modifica la altura minima
 * de acceso a la atraccion
 *
 * @param min Nueva altura minima
 */
public void setMinAlturaCM(float min)
{
    this.minAlturaCM = min;
}

/**
 * Method getMaxAlturaCM
 *
 * Devuelve la altura maxima de acceso a la
 * atraccion en CM
 * Si no hay altura maxima devuelve 0
 *
 * @return Altura maxima en CM
 */
public float getMaxAlturaCM()
{
    return this.maxAlturaCM;
}

/**
 * Method setMaxAlturaCM
 *
 * Modifica la altura maxima de acceso
 * a la atraccion
 *
 * @param max Altura maxima de acceso a la atraccion
 */
public void setMaxAlturaCM(float max)
{
    this.maxAlturaCM = max;
}

/**
 * Method getEdad
 *
 * Devuelve la edad minima de acceso a la atraccion
 * Si no hay edad minima devuelve 0
 *
 * @return Edad minima de acceso a la atraccion
 */
public int getEdad()
{
```

```
    return this.minEdad;
}

/**
 * Method setMinEdad
 *
 * Modifica la edad minima
 * de entrada a la atraccion
 *
 * @param min Edad minima de entrada a la atraccion
 */
public void setMinEdad(int min)
{
    this.minEdad = min;
}

/**
 * Method getAccesoNiños
 *
 * Indica si la atraccion permite
 * el acceso a visitantes con entrada
 * de niño
 *
 * @return Acceso de niños - True si se permite el acceso
 */
public boolean getAccesoNiños()
{
    return this.accesoNiños;
}

/**
 * Method setAccesoAdultos
 *
 * Modifica el acceso a no-niños
 * a la atraccion
 *
 * @param acceso Acceso a no-niños (boolean)
 */
public void setAccesoAdultos(boolean acceso)
{
    this.accesoAdultos= acceso;
}

/**
 * Method getAccesoAdultos
 *
 * Indica si la atraccion permite
 * el acceso de visitantes que no
 * sean niños
 *
```

```
 * @return Acceso de adultos - True si se permite
 */
public boolean getAccesoAdultos()
{
    return this.accesoAdultos;
}

/**
 * Method setAccesoNiños
 *
 * Modifica el acceso a niños
 * de la atraccion
 *
 * @param acceso Acceso a niños (boolean)
 */
public void setAccesoNiños(boolean acceso)
{
    this.accesoNiños= acceso;
}

/**
 * Method setNumRespAtracc
 *
 * Modifica el numero de Responsables
 * de Atraccion necesarios para el uso de la
 * atraccion
 *
 * @param num Numero de Responsables de Atraccion necesarios
 */
public void setNumRespAtracc(int num)
{
    this.numRespAtracc = num;
}

/**
 * Method getNumRespAtracc
 *
 * Devuelve el numero de Responsables
 * de Atraccion necesarios para el uso de la
 * atraccion.
 *
 * @return Numero de Responsables de Atraccion necesarios
 */
public int getNumRespAtracc()
{
    return numRespAtracc;
}

/**
 * Method setNumAyuAtracc
```

```
 *
 * Modifica el numero de Ayudantes de
 * Atraccion necesario para el uso de la
 * atraccion
 *
 * @param num Numero de Ayudantes de Atraccion necesarios
 */
public void setNumAyuAtracc(int num)
{
    numAyuAtracc = num;
}

/**
 * Method getNumAyuAtracc
 *
 * Devuelve el numero de Ayudantes de
 * Atraccion necesario para el uso de la
 * atraccion
 *
 * @return Numero de Ayudantes de Atraccion necesarios
 */
public int getNumAyuAtracc()
{
    return numAyuAtracc;
}

/**
 * Method addTrabajador
 *
 *  Añade un nuevo trabajador a la lista
 *  de trabajadores de la atraccion
 *
 * @param trabajador Nuevo trabajador
 */
public void addTrabajador(Trabajador trabajador)
{
    trabajadores.add(trabajador);
}

/**
 * Method getTrabajadores
 *
 * Devuelve la lista de trabajadores de la atraccion
 *
 * @return Lista de trabajadores de la atraccion
 */
public List getTrabajadores()
{
  return trabajadores;
}
```

```
    /**
     * Method usar
     *
     * Añade un nuevo visitante a la lista de
     * visitantes de la atraccion.
     * Tambien se encarga de verificar que el visitante
     * tenga acceso a la atraccion a partir de su entrada
     *
     * @param entrada Entrada del visitante
     */
    public void usar(EntradaIF entrada)
    {
         contador++;
        if (!(((entrada instanceof Niño) && !getAccesoNiños()) ||
(!(entrada instanceof Niño) && !getAccesoAdultos())))
        {
            usuarios.add(entrada);
            //System.out.println("Acceso permitido a la atraccion");
        }
        else
        {
            System.out.println("Acceso denegado a la atraccion");
        }

    }

    /**
     * Method getUsuarios
     *
     *  Devuelve la lista de visitantes
     *  de la atraccion
     *
     * @return Lista de visitantes de la atraccion
     */
    public List getUsuarios()
    {
      return this.usuarios;
    }
}
```

CreadorAtracciones

```
/**
 * Esta clase se ocupa de la creacion
 * de nuevas atracciones.
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */

import java.util.List;

public class CreadorAtracciones
{
   private AtraccionIF atraccionActual;
    /**
     * Constructor for objects of class CreadorAtracciones
     */
    public CreadorAtracciones()
    {

    }

    /**
     * Method nuevaAtraccion
     *
     * Crea y devuelve una nueva atraccion
     * segun el tipo de atraccion
     * especificado
     *
     * @param tipo Tipo de atraccion (String)
     * @return Nueva atraccion
     */
    public AtraccionIF nuevaAtraccion(String tipo)
    {
        switch(tipo) {
            case "A":
                atraccionActual = new AtraccionA();
                break;
            case "B":
                atraccionActual = new AtraccionB();
                break;
            case "C":
                atraccionActual = new AtraccionC();
                break;
            case "D":
                atraccionActual = new AtraccionD();
                break;
            case "E":
                atraccionActual = new AtraccionE();
```

```
                break;
            default:
                atraccionActual = new AtraccionA();
        }
        return atraccionActual;
    }
}
```

Extensiones de la Clase Abstracta Trabajador

Las clases AyudanteAtraccion, RespAtraccion, RelPublicas y AtencionCl extienden a la clase abstracta Trabajador, implementando dos métodos publicos: getTipo() y getSueldo(). So idénticas aparte de los atributos, y por tanto solo se incluye un ejemplo del código fuente.

```
/**
 * Extiende clase Trabajador para
 * Ayudante de Atraccion.
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */
public class AyudanteAtraccion extends Trabajador
{
     private  TiposTrabajadores tipo;
     private float sueldo;
    /**
     * Constructor for objects of class AtencionCl
     */
    public AyudanteAtraccion()
    {
        super();
        tipo = TiposTrabajadores.AYU_ATRACC;
        sueldo = 31.67f; // sueldo diario
    }


   /**
    * Method getTipo
    *
    * Devuelve el tipo de trabajador
    *
    * @return The return value
    */
   public TiposTrabajadores getTipo()
   {
       return tipo;
   }

   /**
    * Method getSueldo
    *
    * Devuelve el sueldo del trabajador
    *
    * @return The return value
    */
   public float getSueldo()
   {
       return sueldo;
```

```
    }
}
```

CreadorTrabajadores

```
/**
 * Clase que se ocupa de la creacion de nuevos trabajadores.
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */
public class CreadorTrabajadores
{
    // instance variables - replace the example below with your own
    private Trabajador trabajadorActual;

    /**
     * Constructor for objects of class CreadorTrabajadores
     */
    public CreadorTrabajadores()
    {
    }

    /**
     * Method nuevoTrabajador
     *
     * Crea y devuelve un nuevo trabajador
     * segun el tipo de trabajador especificado
     * en el parametro
     *
     * @param tipo Tipo de trabajador (TiposTrabajadores)
     * @return Nuevo trabajador
     */
    public Trabajador nuevoTrabajador(TiposTrabajadores tipo)
    {
        switch (tipo)
        {
            case ATENCION_CL:
              trabajadorActual = new AtencionCl();
              break;
            case RESP_ATRACC:
                trabajadorActual = new RespAtraccion();
                break;
            case AYU_ATRACC:
                trabajadorActual = new AyudanteAtraccion();
                break;
            case REL_PUBLICAS:
                trabajadorActual = new RelPublicas();
                break;
        }

        return trabajadorActual;
    }
```

```
}
```

TiposTrabajadores (enum)

```
/**
 * Esta clase enumera los diferentes tipos de trabajadores
 * del parque
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */
public enum TiposTrabajadores
{
    ATENCION_CL, RESP_ATRACC, AYU_ATRACC, REL_PUBLICAS
}
```

ParqueManager

```
/**
 * Clase principal del programa para almacenar y gestionar
 * los objetos y funcionalidades principales del programa
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */

import java.util.List;
import java.util.LinkedList;
import java.time.*;
import java.util.concurrent.ThreadLocalRandom;

public class ParqueManager

{
    private List<AtraccionIF> AtraccionesParque;
    private List<Trabajador> TrabajadoresParque;
    private List<EntradaIF> EntradasParque;
    private MaquinaEntradas generadorEntradas;
    private CreadorTrabajadores generadorTrabajadores;
    private CreadorAtracciones generadorAtracciones;
    private AtraccionesFuncionando atraccionesFuncionando;
    /**
     * Constructor for objects of class Parque
     */
    public ParqueManager()
    {
        AtraccionesParque = new LinkedList<AtraccionIF>();
        TrabajadoresParque = new LinkedList<Trabajador>();
        EntradasParque = new LinkedList<EntradaIF>();
        generadorEntradas = new MaquinaEntradas();
        generadorTrabajadores = new CreadorTrabajadores();
        generadorAtracciones = new CreadorAtracciones();
        atraccionesFuncionando = new AtraccionesFuncionando();

    }

    /**
     * Method addEntrada
     *
     * Crea una nueva entrada (EntradaGen)
     * y la añade a la lista de entradas
     *
     * @param fecha Fecha de compra de la entrada
     * @param edad Edad del visitante
     * @param isVIP Si la entrada es VIP
```

```
     * @param isFamilia Si es una entrada familias
     * @param descuento Descuentos aplicados
     */
    public void addEntrada(LocalDate fecha, int edad,
                           boolean isVIP, boolean isFamilia,
                           String descuento)
    {
        EntradaIF entrada = generadorEntradas.nuevaEntrada(fecha, edad,
                                                   isVIP, isFamilia,
descuento);
        EntradasParque.add(entrada);
    }

    /**
     * Method addEntradaNiño
     *
     * Crea una nueva entrada tipo niño (Niño)
     * y la añade a la lista de entradas
     *
     * @param fecha Fecha de compra de la entrada
     * @param edad Edad del visitante
     * @param isVIP Si la entrada es VIP
     * @param isFamilia Si es una entrada familias
     * @param descuento Descuentos aplicados
     */
    public void addEntradaNiño(LocalDate fecha, int edad,
                           boolean isVIP, boolean isFamilia,
                           String descuento)
    {
        EntradaIF entrada = generadorEntradas.nuevaEntradaNiño(fecha,
edad,
                                                   isVIP, isFamilia,
descuento);
        EntradasParque.add(entrada);
    }

    /**
     * Method addTrabajador
     *
     * Crea y añade un nuevo trabajador del tipo
     * especificado en el parametro
     *
     * @param tipo Tipo de trabajador (TiposTrabajadores)
     */
    public void addTrabajador(TiposTrabajadores tipo)
    {
        Trabajador trabajador =
generadorTrabajadores.nuevoTrabajador(tipo);
        TrabajadoresParque.add(trabajador);
    }
```

```
    /**
     * Method addAtraccion
     *
     * Crea una nueva atraccion y la añade a la lista
     * de atracciones. Tambien se ocupa de crear los trabajdores
necesarios
     * para el uso de la atraccion y los añade a a la lista de
trabajadores.
     *
     * @param tipo Tipo de atraccion
     */
    public void addAtraccion(String tipo)
    {
        AtraccionIF atraccion = generadorAtracciones.nuevaAtraccion(tipo);
        Trabajador nuevoTrabajador;
        for (int i = 0; i < atraccion.getNumRespAtracc(); i++)
        {
            nuevoTrabajador =
generadorTrabajadores.nuevoTrabajador(TiposTrabajadores.RESP_ATRACC);
            TrabajadoresParque.add(nuevoTrabajador);
            atraccion.addTrabajador(nuevoTrabajador);
        }

        for (int i = 0; i < atraccion.getNumAyuAtracc(); i++)
        {
            nuevoTrabajador =
generadorTrabajadores.nuevoTrabajador(TiposTrabajadores.AYU_ATRACC);
            TrabajadoresParque.add(nuevoTrabajador);
            atraccion.addTrabajador(nuevoTrabajador);
        }

        AtraccionesParque.add(atraccion);
    }

    /**
     * Method getNumTrabajadores
     *
     * Devuelve el numero de trabajadores
     * del parque de un cierto tipo
     *
     * @param tipo Tipo de trabajadors
     * @return Numero de trabajadores
     */
    public int getNumTrabajadores(TiposTrabajadores tipo)
    {
        int n = 0;

        for (Trabajador trabajador : TrabajadoresParque)
        {
```

```
            if (trabajador.getTipo() == tipo)
            {
                n++;
            }
        }

        return n;
    }

    /**
     * Method getNumTrabajadores
     *
     * Devuelve el numero total de trabajadores del parque.
     *
     * @return Total de trabajadores del parque
     */
    public int getNumTrabajadores()
    {
        return TrabajadoresParque.size();
    }

    /**
     * Method getNumVisitantes
     *
     * Devuelve el total de visitantes
     * del parque
     *
     * @return Total de visitantes del parque
     */
    public int getNumVisitantes()
    {
        return EntradasParque.size();
    }

    /**
     * Method usarAtraccion
     *
     * Metodo de apoyo para usar una atraccion
     * especifica
     *
     * @param index Referencia a la atraccion en la lista de atracciones
(indice)
     * @param entrada Entrada del visitante que usa la atraccion
     */
    public void usarAtraccion(int index, EntradaIF entrada)
    {
        AtraccionesParque.get(index).usar(entrada);
    }
```

```
    /**
     * Method randomUsarAtracciones
     *
     * Metodo de apoyo que genera un uso
     * aleatorio de las atracciones del parque
     *
     */
    public void randomUsarAtracciones()
    {
        for (AtraccionIF atraccion : AtraccionesParque)
        {
            int n = ThreadLocalRandom.current().nextInt(10) + 1;
            int i = 0;
            for (EntradaIF entrada: EntradasParque)
            {
                if (i%n == 0 || i%n == 2 || i%n == 5)
                {
                    atraccion.usar(entrada);
                }
                i++;
            }

        }
    }

    /**
     * Method setContenidoAtraccionesFuncionando
     *
     * Genera contenido para el objeto clase AtraccionesFuncionando
     * Que despues se usara en las estadisticas
     * Las atracciones funcionando se turnan en dos periodos (mitades del
año 2019)
     *
     */
    public void setContenidoAtraccionesFuncionando()
    {
      List<AtraccionIF> atraccFuc1 = new LinkedList<AtraccionIF>();
      for (int i = 0; i < AtraccionesParque.size(); i += 2)
      {
        atraccFuc1.add(AtraccionesParque.get(i));
      }
      atraccionesFuncionando.addAtraccion(new
PeriodoTemporada(LocalDate.of(2019, Month.JANUARY, 1), LocalDate.of(2019,
Month.MAY, 31)), atraccFuc1);
      List<AtraccionIF> atraccFuc2 = new LinkedList<AtraccionIF>();
      for (int i = 1; i < AtraccionesParque.size(); i += 2)
      {
        atraccFuc2.add(AtraccionesParque.get(i));
      }
```

```
      atraccionesFuncionando.addAtraccion(new
PeriodoTemporada(LocalDate.of(2019, Month.JUNE, 1), LocalDate.of(2019,
Month.DECEMBER, 31)), atraccFuc2);
    }

    /**
     * Method analisisEstadistico
     *
     * Crea y devuelve un nuevo objeto AnalizadorEstadisticas
     *
     * @return Objeto AnalizadorEstadisticas
     */
    public AnalizadorEstadisticas analisisEstadistico()
    {
        AnalizadorEstadisticas analizador = new
AnalizadorEstadisticas(EntradasParque, AtraccionesParque,
TrabajadoresParque, atraccionesFuncionando);
        return analizador;
    }
}
```

parque (metodo main)

```
/**
 * Clase con el metodo main del sistema.
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */
public class parque
{

    public static void main (String[] args)
    {
        ParqueManager manager = new ParqueManager();

        System.out.println("Bienvenido a Parque de Atracciones PARQUNED");
        System.out.println();
        System.out.println();

        GeneradorVisitantes.generarVisitantes(manager);

        AnalizadorEstadisticas analizador = manager.analisisEstadistico();

        GeneradorContenido.generadorContenido(manager);

        System.out.println("Generando Atracciones....");


        System.out.println("Estructuras Generadas: ");
        System.out.println();
        analizador.resumenAtracciones();
        System.out.println();
        System.out.println("Resumen Trabajadores: ");
        System.out.println();
        System.out.println("Ayudantes de Atraccion: " +
analizador.resumenTrabajadoresTipo(TiposTrabajadores.AYU_ATRACC));
        System.out.println("Responsables de Atraccion: " +
analizador.resumenTrabajadoresTipo(TiposTrabajadores.RESP_ATRACC));
        System.out.println("Atencion al Cliente: " +
analizador.resumenTrabajadoresTipo(TiposTrabajadores.ATENCION_CL));
        System.out.println("Relaciones Publicas: " +
analizador.resumenTrabajadoresTipo(TiposTrabajadores.REL_PUBLICAS));
        System.out.println();
        System.out.println("Importando Visitantes del archivo
Visitantes.txt ...");
        System.out.println();
        analizador.resumenVisitantesTipo();
        System.out.println("====================================");
    }
}
```

Tarea 3:

AnalizadorEstadisticas

```
/**
 * Write a description of class AnalizadorEstadisticas here.
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */

import java.util.List;
import java.util.LinkedList;
import java.util.HashMap;
import java.time.*;
import java.time.temporal.*;
import java.util.Locale;
import java.math.BigDecimal;

public class AnalizadorEstadisticas
{
   private List<EntradaIF> ListaEntradas;
   private List<AtraccionIF> ListaAtracciones;
   private List<Trabajador> ListaTrabajadores;
   private AtraccionesFuncionando atraccionesActivas;

   public AnalizadorEstadisticas(List<EntradaIF> listaEntradas, List<AtraccionIF> listaAtracc,
List<Trabajador> listaTrab, AtraccionesFuncionando atraccionesAct)
   {
      ListaEntradas = listaEntradas;
      ListaAtracciones = listaAtracc;
      ListaTrabajadores = listaTrab;
      this.atraccionesActivas = atraccionesAct;
   }

   /**
    * Metodo resumenAtracciones
    *
    * Imprime en pantalla un recuento de las
    * atracciones creadas en el parque
    *
    */
   public void resumenAtracciones()
   {
      HashMap<String, Integer> tiposAtracciones = new HashMap();
```

```
    for (AtraccionIF atraccion : ListaAtracciones)
    {
      String tipo = atraccion.getTipo();
      if (tiposAtracciones.containsKey(tipo))
      {
        tiposAtracciones.put(tipo, tiposAtracciones.get(tipo) + 1);
      }
      else
      {
        tiposAtracciones.put(tipo, 1);
      }
    }
    for (String tipo : tiposAtracciones.keySet())
    {
      System.out.println("Atracciones " + tipo + ": " + tiposAtracciones.get(tipo));
    }
  }

  /**
   * Method resumenAtraccionesActivas
   *
   * Metodo para mostrar informacion
   * sobre las atracciones activas
   *
   */
  public void resumenAtraccionesActivas()
  {
    atraccionesActivas.imprPeriodos();
  }


/**
 * Metodo resumenTrabajadoresTipo
 *
 * Cuenta el numero de trabajadores del tipo
 * especificado y lo devuelve como entero
 *
 * @param tipo Tipo de trabajador (TiposTrabajadores)
 * @return Numero de trabajadores del tipo especificado (entero)
 */
public int resumenTrabajadoresTipo(TiposTrabajadores tipo)
{
    int n = 0;
```

```
      for (Trabajador trabajador : ListaTrabajadores)
      {
         if (trabajador.getTipo() == tipo)
         {
            n++;
         }
      }

      return n;
   }

   /**
    * Method trabajadoresPorAtraccion
    *
    * Metodo para mostrar informacion
    * sobre los trabajadores por atraccion
    * del parque
    *
    */
   public void trabajadoresPorAtraccion()
   {
     int counter = 0;
     for (AtraccionIF atraccion : ListaAtracciones)
     {
      List<Trabajador> trabajadores = new
LinkedList<Trabajador>(atraccion.getTrabajadores());
      HashMap<TiposTrabajadores, Integer> contadorTrabajadores = new HashMap();
      System.out.println("Atraccion #" + counter + " tipo " + atraccion.getTipo() + ": ");
      counter++;

      for (Trabajador trabajador : trabajadores)
      {
       if (!contadorTrabajadores.containsKey(trabajador.getTipo()))
       {
          contadorTrabajadores.put(trabajador.getTipo(), 1);
       }
       else
       {
        contadorTrabajadores.put(trabajador.getTipo(),
(contadorTrabajadores.get(trabajador.getTipo()) + 1));
       }
      }
      for (TiposTrabajadores tipo : contadorTrabajadores.keySet())
      {
```

```
        switch (tipo)
        {
          case AYU_ATRACC:
            System.out.println("Ayudantes de Atraccion: " + contadorTrabajadores.get(tipo));
            break;
          case RESP_ATRACC:
            System.out.println("Responsables de Atraccion: " + contadorTrabajadores.get(tipo));
            break;
        }
       }
        System.out.println();
     }
     System.out.println("Relaciones Publicas: " +
resumenTrabajadoresTipo(TiposTrabajadores.REL_PUBLICAS));
     System.out.println("Atencion al Cliente: " +
resumenTrabajadoresTipo(TiposTrabajadores.ATENCION_CL));

     System.out.println();
   }

   /**
    * Metodo resumenVisitantesTipo
    *
    * Imprime en pantalla un recuento de los visitantes
    * del parque agrupados por tipo/descuento de la entrada
    *
    */
   public void resumenVisitantesTipo()
   {
    HashMap<String, Integer> tiposVisitantes = new HashMap();
    for (EntradaIF entrada : ListaEntradas)
    {
     String tipo = entrada.getTipo();
     if (tiposVisitantes.containsKey(tipo))
     {
      tiposVisitantes.put(tipo, tiposVisitantes.get(tipo) + 1);
     }
     else
     {
      tiposVisitantes.put(tipo, 1);
     }
    }
    for (String tipo : tiposVisitantes.keySet())
    {
```

```
    System.out.println(tipo + ": " + tiposVisitantes.get(tipo));
  }
}


/**
 * Method resumenVisitantes
 *
 * Metodo para generar estadisticas sobre
 * los visitantes del parque en un año
 *
 * @param year Año
 */
public void resumenVisitantes(int year)
{
   analisisPorFechas(ListaEntradas, year);
}

/**
 * Method promedioSemanal
 *
 * Calcula un promedio semanal de un entero
 * (lo divide por siete)
 *
 * @param n Entero del que se quiere calcular el promedio semanal
 * @return Promedio semanal (float)
 */
private float promedioSemanal(int n)
{
   float promedio = (float) (n*1.0f/7);
   return promedio;
}

/**
 * Method promedioMensual
 *
 * Calcula el promedio mensual de un entero
 * dividiendolo por el numero de dias del mes
 *
 * @param n Entero del que se quiere calcular el promedio mensual
 * @param m Numero de dias del mes
 * @return Promedio mensual (float)
 */
private float promedioMensual(int n, int m)
```

```
{
   float promedio = (float) ((n*1.0f)/m);
   return promedio;
}

/**
 * Method promedioAnual
 *
 * Calcula el promedio anual de un entero,
 * dividiendolo por el numero de dias de un año natural
 *
 * @param n Entero del que se quiere calcular el promedio anual
 * @return Promedio anual
 */
private float promedioAnual(int n)
{
   float promedio = (float) ((n*1.0f)/365);
   return promedio;
}

//RESUMEN PRECIOS


/**
 * Method resumenPrecios
 *
 * Metodo para generar estadisticas sobre los
 * precios de las entradas en el periodo de un
 * añp
 *
 * @param year Año
 */
public void resumenPrecios(int year)
{
   int dia = ListaEntradas.get(0).getDate().getDayOfMonth();
   int month = ListaEntradas.get(0).getDate().getMonthValue();
   int daysInMonth = ListaEntradas.get(0).getDate().lengthOfMonth();
   TemporalField woy = WeekFields.of(Locale.getDefault()).weekOfWeekBasedYear();
   int semana = ListaEntradas.get(0).getDate().get(woy);
   int anno = year;

   int contadorDia = 0;
   int contadorSemana = 0;
   int contadorMes = 0;
```

```
int contadorAnno = 0;

float sumadorDia = 0;
float sumadorSemana = 0;
float sumadorMes = 0;
float sumadorAnno = 0;

float promedioImpSemanal = 0;
float promedioImpMes = 0;
float promedioImpAnno = 0;

LocalDate ultimaFecha = ListaEntradas.get(ListaEntradas.size()-1).getDate();

System.out.println("Año: " + anno);
System.out.println("Mes: " + month);
System.out.println("  Semana: " + semana);

for (EntradaIF entrada : ListaEntradas)
{
   LocalDate fecha = entrada.getDate();
   if (fecha.getYear() == anno)
   {
      if (fecha.getMonthValue() == month) //Loop para el mes
      {
         if (fecha.get(woy) == semana)
         {
            if (fecha.getDayOfMonth() == dia)
            {
               contadorDia++;
               contadorSemana++;
               contadorMes++;
               contadorAnno++;

               sumadorDia += entrada.getPrecio();
               sumadorSemana += entrada.getPrecio();
               sumadorMes += entrada.getPrecio();
               sumadorAnno += entrada.getPrecio();
            }
            else
            {
               System.out.println("   " + dia + "/" +
                                  month + "/" +
                                  anno +
```

```
                        " - Precio Medio: $" + promedioPrecio(sumadorDia,
contadorDia));
                    contadorDia = 1;
                    sumadorDia = entrada.getPrecio();
                    dia = fecha.getDayOfMonth();

                    contadorSemana++;
                    sumadorSemana += entrada.getPrecio();
                    contadorMes++;
                    sumadorMes += entrada.getPrecio();
                    contadorAnno++;
                    sumadorAnno += entrada.getPrecio();

                  }
                }
                else
                {
                  System.out.println("   " + dia + "/" +
                                month + "/" +
                                anno +
                                " - Precio Medio: $" + promedioPrecio(sumadorDia, contadorDia));
                contadorDia = 1;
                dia = fecha.getDayOfMonth();

                System.out.println("    Total Semana: $" + sumadorSemana);
                System.out.println("\t\t\t\t    Promedio Semanal: $" +
promedioPrecio(sumadorSemana, contadorSemana));
                contadorSemana = 1;
                sumadorSemana = entrada.getPrecio();
                semana = fecha.get(woy);
                System.out.println("  Semana: " + semana);

                contadorMes++;
                sumadorMes += entrada.getPrecio();
                contadorAnno++;
                sumadorAnno += entrada.getPrecio();
                }
              }
              else
              {
                System.out.println("   " + dia + "/" +
                                month + "/" +
                                anno +
                                " - Precio Medio: $" + promedioPrecio(sumadorDia, contadorDia));
```

```
            contadorDia = 1;
            dia = fecha.getDayOfMonth();
            month = fecha.getMonthValue();
            if (fecha.get(woy) != semana)//Casos en los que fin de semana y fin de mes
coinciden
            {
               System.out.println("    Total Semana: $" + sumadorSemana);
               System.out.println("\t\t\t\t    Promedio Semanal: $" +
promedioPrecio(sumadorSemana, contadorSemana));
               contadorSemana = 1;
               sumadorSemana = entrada.getPrecio();
               semana = fecha.get(woy);

               System.out.println("  Total Mes: $" + sumadorMes);
               System.out.println("\t\t\t\t    Promedio Mensual: $" +
promedioPrecio(sumadorMes, contadorMes));
               contadorMes = 1;
               sumadorMes = entrada.getPrecio();
               contadorAnno++;
               sumadorAnno += entrada.getPrecio();

               System.out.println("Mes: " + month);
               System.out.println("  Semana: " + semana);
            }
            else
            {
               contadorSemana++;
               sumadorSemana += entrada.getPrecio();

               System.out.println("  Total Mes: $" + sumadorMes);
               System.out.println("\t\t\t\t    Promedio Mensual:$" +
promedioPrecio(sumadorMes, contadorMes));
               contadorMes = 1;
               sumadorMes = entrada.getPrecio();
               System.out.println("Mes: $" + month);

               contadorAnno++;
               sumadorAnno += entrada.getPrecio();
            }
            daysInMonth = fecha.lengthOfMonth();

         }
      }
```

```
        if (fecha == ultimaFecha || fecha.getYear() > year)
        {
           dia = fecha.getDayOfMonth();
           month = fecha.getMonthValue();
           daysInMonth = fecha.lengthOfMonth();

           System.out.println("   " + dia + "/" +
                              month + "/" +
                              fecha.getYear() +
                              " - Precio Medio: $" + promedioPrecio(sumadorDia, contadorDia));

           System.out.println("    Total Semana: " + sumadorSemana);
           System.out.println("\t\t\t\t    Promedio Semanal: $" +
promedioPrecio(sumadorSemana, contadorSemana));

           System.out.println("  Total Mes: " + sumadorMes);
           System.out.println("\t\t\t\t    Promedio Mensual: $" +
promedioPrecio(sumadorMes, contadorMes));

           System.out.println(" Total Año: " + sumadorAnno);
           System.out.println("\t\t\t\t    Promedio Anual: $" + promedioPrecio(sumadorAnno,
contadorAnno));
        }

     }
  }

  /**
   * Method promedioPrecio
   *
   * Funcion para calcular un promedio, dividiendo por n
   *
   * @param imp Float del que se quiere calcular el promedio
   * @param n Entero por el que se desea dividir
   * @return Promedio de n / m
   */
  private float promedioPrecio (float imp, int n)
  {
    float promedio = (float) (imp*1.0f)/n;
    return round(promedio, 2);
  }
```

```
/**
 * Method resumenVisitasAtracciones
 *
 * Metodo para genera estadisticas sobre
 * el uso de una atraccion en el periodo
 * de un año
 *
 * @param year año
 * @param atraccion Atraccion
 */
public void resumenVisitasAtracciones(int year, AtraccionIF atraccion)
{

   System.out.println("Atraccion - Tipo " + atraccion.getTipo());
   List<EntradaIF> usuarios = atraccion.getUsuarios();

   analisisPorFechas(usuarios, year);

  }

/**
 * Method analisisPorFechas
 *
 * Funcion base para generar las estadisticas sobre entradas
 * agrupadas por dias, semana, mes y año. Codigo se puede
 * reutilizar, con ligeras modificaciones, para otros
 * tipos de lista
 *
 * @param listaEntradas A parameter
 * @param year Año
 */
private void analisisPorFechas(List<EntradaIF> listaEntradas, int year)
{
  int dia = listaEntradas.get(0).getDate().getDayOfMonth();
  int month = listaEntradas.get(0).getDate().getMonthValue();
  int daysInMonth = listaEntradas.get(0).getDate().lengthOfMonth();
  TemporalField woy = WeekFields.of(Locale.getDefault()).weekOfWeekBasedYear();
  int semana = listaEntradas.get(0).getDate().get(woy);
  int anno = year;

  int contadorDia = 0;
  int contadorSemana = 0;
  int contadorMes = 0;
```

```
int contadorAnno = 0;

float promedioSemanal = 0;
float promedioMes = 0;
float promedioAnno = 0;

LocalDate ultimaFecha = listaEntradas.get(listaEntradas.size()-1).getDate();

System.out.println("Año: " + anno);
System.out.println("Mes: " + month);
System.out.println("  Semana: " + semana);

for (EntradaIF entrada : listaEntradas)
{
   LocalDate fecha = entrada.getDate();
   if (fecha.getYear() == anno)
   {
      if (fecha.getMonthValue() == month) //Loop para el mes
      {
         if (fecha.get(woy) == semana)
         {
            if (fecha.getDayOfMonth() == dia)
            {
               contadorDia++;
               contadorSemana++;
               contadorMes++;
               contadorAnno++;
            }
            else
            {
               System.out.println("   " + dia + "/" +
                                  month + "/" +
                                  anno +
                                  " - Total Visitantes: " + contadorDia);
               contadorDia = 1;
               dia = fecha.getDayOfMonth();

               contadorSemana++;
               contadorMes++;
               contadorAnno++;
            }
         }
         else
         {
```

```
            System.out.println("   " + dia + "/" +
                               month + "/" +
                               anno +
                               " - Total Visitantes: " + contadorDia);
            contadorDia = 1;
            dia = fecha.getDayOfMonth();

            System.out.println("    Total Semana: " + contadorSemana);
            promedioSemanal = promedioSemanal(contadorSemana);
            System.out.println("\t\t\t\t    Promedio Semanal: " + contadorSemana);
            contadorSemana = 1;
            semana = fecha.get(woy);
            System.out.println("  Semana: " + semana);

            contadorMes++;
            contadorAnno++;
            }
         }
         else
         {
            System.out.println("   " + dia + "/" +
                               month + "/" +
                               anno +
                               " - Total Visitantes: " + contadorDia);
            contadorDia = 1;
            dia = fecha.getDayOfMonth();
            month = fecha.getMonthValue();
            if (fecha.get(woy) != semana)//Casos en los que fin de semana y fin de mes
coinciden
            {
               System.out.println("    Total Semana: " + contadorSemana);
               promedioSemanal = promedioSemanal(contadorSemana);
               System.out.println("\t\t\t\t    Promedio Semanal: " + contadorSemana);
               contadorSemana = 1;
               semana = fecha.get(woy);

               System.out.println("  Total Mes: " + contadorMes);
               promedioMes = promedioMensual(contadorMes, daysInMonth);
               System.out.println("\t\t\t\t    Promedio Mensual: " + promedioMes);
               contadorMes = 1;
               contadorAnno++;

               System.out.println("Mes: " + month);
               System.out.println("  Semana: " + semana);
```

```
            }
            else
            {
               contadorSemana++;

               System.out.println("  Total Mes: " + contadorMes);
               promedioMes = promedioMensual(contadorMes, daysInMonth);
               System.out.println("\t\t\t\t    Promedio Mensual: " + promedioMes);
               contadorMes = 1;
               System.out.println("Mes: " + month);

               contadorAnno++;
            }
            daysInMonth = fecha.lengthOfMonth();

         }
      }


      if (fecha == ultimaFecha || fecha.getYear() > year)
      {
         dia = fecha.getDayOfMonth();
         month = fecha.getMonthValue();
         daysInMonth = fecha.lengthOfMonth();

         System.out.println("   " + dia + "/" +
                            month + "/" +
                            fecha.getYear() +
                            " - Total Visitantes: " + contadorDia);

         System.out.println("    Total Semana: " + contadorSemana);
         promedioSemanal = promedioSemanal(contadorSemana);
         System.out.println("\t\t\t\t    Promedio Semanal: " + contadorSemana);

         System.out.println("  Total Mes: " + contadorMes);
         promedioMes = promedioMensual(contadorMes, daysInMonth);
         System.out.println("\t\t\t\t    Promedio Mensual: " + promedioMes);

         System.out.println(" Total Año: " + contadorAnno);
         promedioAnno = promedioAnual(contadorAnno);
         System.out.println("\t\t\t\t    Promedio Anual: " + promedioAnno);
      }
   }
}
```

```
/**
     * Method round
     *
     * Metodo para redondear valores float a dos decimales
     *
     * @param d Valor a redondear (float)
     * @param decimalPlace Numero de decimales (entero)
     * @return Valor redondeado (float)
     */
    private float round(float d, int decimalPlace)
    {
        BigDecimal n = new BigDecimal(Float.toString(d));
        n = n.setScale(decimalPlace, BigDecimal.ROUND_HALF_UP);
        return n.floatValue();
    }
```

parque (método main)

```
/**
 * Clase con el metodo main del sistema.
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */
public class parque
{

    public static void main (String[] args)
    {
        ParqueManager manager = new ParqueManager();

        System.out.println("Bienvenido a Parque de Atracciones PARQUNED");
        System.out.println();
        System.out.println();

        GeneradorVisitantes.generarVisitantes(manager);

        AnalizadorEstadisticas analizador = manager.analisisEstadistico();

        GeneradorContenido.generadorContenido(manager);

        System.out.println("Generando Atracciones....");


        System.out.println("Estructuras Generadas: ");
        System.out.println();
        analizador.resumenAtracciones();
        System.out.println();
        System.out.println("Resumen Trabajadores: ");
        System.out.println();
        System.out.println("Ayudantes de Atraccion: " +
analizador.resumenTrabajadoresTipo(TiposTrabajadores.AYU_ATRACC));
        System.out.println("Responsables de Atraccion: " +
analizador.resumenTrabajadoresTipo(TiposTrabajadores.RESP_ATRACC));
        System.out.println("Atencion al Cliente: " +
analizador.resumenTrabajadoresTipo(TiposTrabajadores.ATENCION_CL));
        System.out.println("Relaciones Publicas: " +
analizador.resumenTrabajadoresTipo(TiposTrabajadores.REL_PUBLICAS));
        System.out.println();
        System.out.println("Importando Visitantes del archivo
Visitantes.txt ...");
        System.out.println();
        analizador.resumenVisitantesTipo();
        System.out.println("====================================");
        System.out.println();
```

```
        System.out.println("Generando Estadisticas...");
        System.out.println();
        System.out.println("Analisis de los Visitantes del Parque");
        analizador.resumenVisitantes();
        System.out.println("====================================");
        System.out.println();
        System.out.println("Analisis del Precio de las Entradas");
        analizador.resumenPrecios();
        System.out.println("====================================");
        System.out.println();
        System.out.println("Generando visitas aleatorias a las
atracciones...");
        System.out.println("Analisis del Uso de las Atracciones");
        analizador.resumenVisitasAtracciones();


    }
}
```

Tarea 4

AtraccionesFuncionando

```
/**
 * Clase AtraccionesFuncionando, que almacena y suministra
 * las atracciones activas en determinadas fechas del año.
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */

import java.util.List;
import java.util.LinkedList;
import java.util.HashMap;
import java.time.*;

public class AtraccionesFuncionando
{

    private HashMap<PeriodoTemporada, List<AtraccionIF> > atraccionesActivas;

    public AtraccionesFuncionando()
    {
        atraccionesActivas = new HashMap<PeriodoTemporada, List<AtraccionIF> >();
    }


  /**
   * Metodo addAtraccion
   *
   * Metodo para añadir listas de atracciones
   * activas para un determinado periodo del año
   *
   * @param periodo Periodo en que las atracciones de la lista estaran activas
   * @param atracciones Lista de atracciones activas durante dicho periodo
   */
  public void addAtraccion(PeriodoTemporada periodo, List<AtraccionIF> atracciones)
  {
    PeriodoTemporada key = buscarPeriodo(periodo);
    if (key != null)
    {
      for (AtraccionIF atraccion : atracciones)
      {
        List<AtraccionIF>              nuevaLista              =              new
LinkedList<AtraccionIF>(atraccionesActivas.get(periodo));
        nuevaLista.add(atraccion);
        atraccionesActivas.put(periodo, nuevaLista);
      }
    }
    else
    {
      if (!buscarCoincidencias(periodo))
      {
        atraccionesActivas.put(periodo, atracciones);
```

```
    }
    else
    {
        System.out.println("Error: Coincidencia de fechas");
    }

  }
}

/**
 * Metodo getAtracciones
 *
 * Metodo que devuelve una lista de atracciones activas
 * en la fecha dada.
 *
 * @param fecha Fecha en la que se buscan atracciones activas
 * @return Lista de atracciones activas para dicha fecha
 */
public List<AtraccionIF> getAtracciones(LocalDate fecha)
{
  for (PeriodoTemporada periodo : atraccionesActivas.keySet())
  {
    if (periodo.enPeriodo(fecha)) {return atraccionesActivas.get(periodo);}
  }
  return new LinkedList<AtraccionIF>();
}

/**
 * Metodo buscarPeriodo
 *
 * Metodo privado para verificar si un periodo dado existe
 *
 * @param periodo Periodo que se debe buscar en el HashMap
 * @return True: si periodo encontrada; False: si periodo no existen en el HashMap
 */
private PeriodoTemporada buscarPeriodo(PeriodoTemporada periodo)
{
  if (!atraccionesActivas.isEmpty())
  {
    for (PeriodoTemporada key : atraccionesActivas.keySet())
    {
      if (key.equals(periodo))
      {
        return key;
      }
    }
    return null;
  }
  else
  {
    return null;
  }
}

/**
```

```
   * Metodo buscarCoincidencias
   *
   * Metodo para verificar si periodo dado coincide con
   * otros periodos en existentes en el HashMap
   *
   * @param periodo Periodo para la busqueda
   * @return True: Si existe coincidendia; False: Si no se encuentra coincidencia
   */
  private boolean buscarCoincidencias(PeriodoTemporada periodo)
  {
    for (PeriodoTemporada key : atraccionesActivas.keySet())
    {
      if (key.enPeriodo(periodo.getFechaInic())
          || key.enPeriodo(periodo.getFechaFinal())
          ||  ((key.getFechaInic().compareTo(periodo.getFechaInic())  <  0)  &&
(key.getFechaFinal().compareTo(periodo.getFechaFinal()) > 0)))
          {
            return true;
          }
    }
    return false;
  }


}
```

AnalizadorEstadisticas (Expansión)

```
/**
 * Write a description of class AnalizadorEstadisticas here.
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */

import java.util.List;
import java.util.LinkedList;
import java.util.HashMap;
import java.time.*;
import java.time.temporal.*;
import java.util.Locale;
import java.math.BigDecimal;

public class AnalizadorEstadisticas
{
    private List<EntradaIF> ListaEntradas;
    private List<AtraccionIF> ListaAtracciones;
    private List<Trabajador> ListaTrabajadores;
    private AtraccionesFuncionando atraccionesActivas;

    public AnalizadorEstadisticas(List<EntradaIF> listaEntradas, List<AtraccionIF>
listaAtracc, List<Trabajador> listaTrab, AtraccionesFuncionando atraccionesAct)
    {
        ListaEntradas = listaEntradas;
        ListaAtracciones = listaAtracc;
        ListaTrabajadores = listaTrab;
        this.atraccionesActivas = atraccionesAct;
    }
```

```
    /**
     * Metodo resumenAtracciones
     *
     * Imprime en pantalla un recuento de las
     * atracciones creadas en el parque
     *
     */
    public void resumenAtracciones()
    {
        HashMap<String, Integer> tiposAtracciones = new HashMap();
        for (AtraccionIF atraccion : ListaAtracciones)
        {
          String tipo = atraccion.getTipo();
          if (tiposAtracciones.containsKey(tipo))
          {
            tiposAtracciones.put(tipo, tiposAtracciones.get(tipo) + 1);
          }
          else
          {
            tiposAtracciones.put(tipo, 1);
          }
        }
        for (String tipo : tiposAtracciones.keySet())
        {
          System.out.println("Atracciones    "    +    tipo    +    ":    "    +
tiposAtracciones.get(tipo));
        }
      }


      /**
       * Method resumenAtraccionesActivas
       *
       * Metodo para mostrar informacion
       * sobre las atracciones activas
```

```
   *
   */
  public void resumenAtraccionesActivas()
  {
    atraccionesActivas.imprPeriodos();
  }


/**
 * Metodo resumenTrabajadoresTipo
 *
 * Cuenta el numero de trabajadores del tipo
 * especificado y lo devuelve como entero
 *
 * @param tipo Tipo de trabajador (TiposTrabajadores)
 * @return Numero de trabajadores del tipo especificado (entero)
 */
public int resumenTrabajadoresTipo(TiposTrabajadores tipo)
{
    int n = 0;

    for (Trabajador trabajador : ListaTrabajadores)
    {
        if (trabajador.getTipo() == tipo)
        {
            n++;
        }
    }

    return n;
}

/**
 * Method trabajadoresPorAtraccion
```

```
     *
     * Metodo para mostrar informacion
     * sobre los trabajadores por atraccion
     * del parque
     *
     */
    public void trabajadoresPorAtraccion()
    {
      int counter = 0;
      for (AtraccionIF atraccion : ListaAtracciones)
      {
        List<Trabajador> trabajadores = new
LinkedList<Trabajador>(atraccion.getTrabajadores());
        HashMap<TiposTrabajadores, Integer> contadorTrabajadores = new HashMap();
        System.out.println("Atraccion #" + counter + " tipo " + atraccion.getTipo()
+ ": ");
        counter++;

        for (Trabajador trabajador : trabajadores)
        {
          if (!contadorTrabajadores.containsKey(trabajador.getTipo()))
          {
              contadorTrabajadores.put(trabajador.getTipo(), 1);
          }
          else
          {
            contadorTrabajadores.put(trabajador.getTipo(),
(contadorTrabajadores.get(trabajador.getTipo()) + 1));
          }
        }
        for (TiposTrabajadores tipo : contadorTrabajadores.keySet())
        {
          switch (tipo)
          {
            case AYU_ATRACC:
```

```
              System.out.println("Ayudantes      de      Atraccion:      "      +
contadorTrabajadores.get(tipo));
              break;
            case RESP_ATRACC:
              System.out.println("Responsables     de     Atraccion:     "     +
contadorTrabajadores.get(tipo));
              break;
          }
        }
          System.out.println();
      }
      System.out.println("Relaciones           Publicas:           "           +
resumenTrabajadoresTipo(TiposTrabajadores.REL_PUBLICAS));
      System.out.println("Atencion         al         Cliente:         "         +
resumenTrabajadoresTipo(TiposTrabajadores.ATENCION_CL));


      System.out.println();
    }


    /**
     * Metodo resumenVisitantesTipo
     *
     * Imprime en pantalla un recuento de los visitantes
     * del parque agrupados por tipo/descuento de la entrada
     *
     */
    public void resumenVisitantesTipo()
    {
      HashMap<String, Integer> tiposVisitantes = new HashMap();
      for (EntradaIF entrada : ListaEntradas)
      {
        String tipo = entrada.getTipo();
        if (tiposVisitantes.containsKey(tipo))
        {
          tiposVisitantes.put(tipo, tiposVisitantes.get(tipo) + 1);
```

```
    }
    else
    {
      tiposVisitantes.put(tipo, 1);
    }
  }
  for (String tipo : tiposVisitantes.keySet())
  {
    System.out.println(tipo + ": " + tiposVisitantes.get(tipo));
  }
}



/**
 * Method resumenVisitantes
 *
 * Metodo para generar estadisticas sobre
 * los visitantes del parque en un año
 *
 * @param year Año
 */
public void resumenVisitantes(int year)
{
    analisisPorFechas(ListaEntradas, year);
}

/**
 * Method promedioSemanal
 *
 * Calcula un promedio semanal de un entero
 * (lo divide por siete)
 *
 * @param n Entero del que se quiere calcular el promedio semanal
 * @return Promedio semanal (float)
```

```
 */
private float promedioSemanal(int n)
{
    float promedio = (float) (n*1.0f/7);
    return promedio;
}

/**
 * Method promedioMensual
 *
 * Calcula el promedio mensual de un entero
 * dividiendolo por el numero de dias del mes
 *
 * @param n Entero del que se quiere calcular el promedio mensual
 * @param m Numero de dias del mes
 * @return Promedio mensual (float)
 */
private float promedioMensual(int n, int m)
{
    float promedio = (float) ((n*1.0f)/m);
    return promedio;
}

/**
 * Method promedioAnual
 *
 * Calcula el promedio anual de un entero,
 * dividiendolo por el numero de dias de un año natural
 *
 * @param n Entero del que se quiere calcular el promedio anual
 * @return Promedio anual
 */
private float promedioAnual(int n)
{
```

```
        float promedio = (float) ((n*1.0f)/365);
        return promedio;
    }

    //RESUMEN PRECIOS



    /**
     * Method resumenPrecios
     *
     * Metodo para generar estadisticas sobre los
     * precios de las entradas en el periodo de un
     * añp
     *
     * @param year Año
     */
    public void resumenPrecios(int year)
    {
        int dia = ListaEntradas.get(0).getDate().getDayOfMonth();
        int month = ListaEntradas.get(0).getDate().getMonthValue();
        int daysInMonth = ListaEntradas.get(0).getDate().lengthOfMonth();
        TemporalField                                woy                                =
WeekFields.of(Locale.getDefault()).weekOfWeekBasedYear();
        int semana = ListaEntradas.get(0).getDate().get(woy);
        int anno = year;


        int contadorDia = 0;
        int contadorSemana = 0;
        int contadorMes = 0;
        int contadorAnno = 0;


        float sumadorDia = 0;
        float sumadorSemana = 0;
        float sumadorMes = 0;
```

```
        float sumadorAnno = 0;


        float promedioImpSemanal = 0;
        float promedioImpMes = 0;
        float promedioImpAnno = 0;


        LocalDate     ultimaFecha     =     ListaEntradas.get(ListaEntradas.size()-
1).getDate();


        System.out.println("Año: " + anno);
        System.out.println("Mes: " + month);
        System.out.println("  Semana: " + semana);


        for (EntradaIF entrada : ListaEntradas)
        {
            LocalDate fecha = entrada.getDate();
            if (fecha.getYear() == anno)
            {
                if (fecha.getMonthValue() == month) //Loop para el mes
                {
                    if (fecha.get(woy) == semana)
                    {
                         if (fecha.getDayOfMonth() == dia)
                        {
                            contadorDia++;
                            contadorSemana++;
                            contadorMes++;
                            contadorAnno++;


                            sumadorDia += entrada.getPrecio();
                            sumadorSemana += entrada.getPrecio();
                            sumadorMes += entrada.getPrecio();
                            sumadorAnno += entrada.getPrecio();
                        }
```

```
                        else
                        {
                            System.out.println("   " + dia + "/" +
                                               month + "/" +
                                               anno +
                                               "    -    Precio    Medio:    $"    +
promedioPrecio(sumadorDia, contadorDia));
                            contadorDia = 1;
                            sumadorDia = entrada.getPrecio();
                            dia = fecha.getDayOfMonth();

                            contadorSemana++;
                            sumadorSemana += entrada.getPrecio();
                            contadorMes++;
                            sumadorMes += entrada.getPrecio();
                            contadorAnno++;
                            sumadorAnno += entrada.getPrecio();

                        }
                    }
                    else
                    {
                        System.out.println("   " + dia + "/" +
                                           month + "/" +
                                           anno +
                                           "     -     Precio     Medio:     $"     +
promedioPrecio(sumadorDia, contadorDia));
                    contadorDia = 1;
                    dia = fecha.getDayOfMonth();

                    System.out.println("    Total Semana: $" + sumadorSemana);
                    System.out.println("\t\t\t\t         Promedio  Semanal:  $"  +
promedioPrecio(sumadorSemana, contadorSemana));
                    contadorSemana = 1;
                    sumadorSemana = entrada.getPrecio();
```

```
                    semana = fecha.get(woy);
                    System.out.println("  Semana: " + semana);

                    contadorMes++;
                    sumadorMes += entrada.getPrecio();
                    contadorAnno++;
                    sumadorAnno += entrada.getPrecio();
                    }
                }
                else
                {
                    System.out.println("   " + dia + "/" +
                                       month + "/" +
                                       anno +
                                       "    -    Precio    Medio:    $"    +
promedioPrecio(sumadorDia, contadorDia));
                    contadorDia = 1;
                    dia = fecha.getDayOfMonth();
                    month = fecha.getMonthValue();
                    if (fecha.get(woy) != semana)//Casos en los que fin de semana
y fin de mes coinciden
                    {
                        System.out.println("    Total Semana: $" + sumadorSemana);
                        System.out.println("\t\t\t\t      Promedio Semanal: $" +
promedioPrecio(sumadorSemana, contadorSemana));
                        contadorSemana = 1;
                        sumadorSemana = entrada.getPrecio();
                        semana = fecha.get(woy);

                        System.out.println("  Total Mes: $" + sumadorMes);
                        System.out.println("\t\t\t\t      Promedio Mensual: $" +
promedioPrecio(sumadorMes, contadorMes));
                        contadorMes = 1;
                        sumadorMes = entrada.getPrecio();
                        contadorAnno++;
```

```
                    sumadorAnno += entrada.getPrecio();

                    System.out.println("Mes: " + month);
                    System.out.println("  Semana: " + semana);
                }
                else
                {
                    contadorSemana++;
                    sumadorSemana += entrada.getPrecio();

                    System.out.println("  Total Mes: $" + sumadorMes);
                    System.out.println("\t\t\t\t       Promedio  Mensual:$"  +
promedioPrecio(sumadorMes, contadorMes));
                    contadorMes = 1;
                    sumadorMes = entrada.getPrecio();
                    System.out.println("Mes: $" + month);

                    contadorAnno++;
                    sumadorAnno += entrada.getPrecio();
                }
                daysInMonth = fecha.lengthOfMonth();

            }
        }



        if (fecha == ultimaFecha || fecha.getYear() > year)
        {
            dia = fecha.getDayOfMonth();
            month = fecha.getMonthValue();
            daysInMonth = fecha.lengthOfMonth();

            System.out.println("   " + dia + "/" +
```

```
                                        month + "/" +
                                        fecha.getYear() +
                                        "     -     Precio     Medio:     $"     +
promedioPrecio(sumadorDia, contadorDia));


                System.out.println("    Total Semana: " + sumadorSemana);
                System.out.println("\t\t\t\t           Promedio   Semanal:   $"   +
promedioPrecio(sumadorSemana, contadorSemana));


                System.out.println("  Total Mes: " + sumadorMes);
                System.out.println("\t\t\t\t           Promedio   Mensual:   $"   +
promedioPrecio(sumadorMes, contadorMes));


                System.out.println(" Total Año: " + sumadorAnno);
                System.out.println("\t\t\t\t            Promedio   Anual:   $"   +
promedioPrecio(sumadorAnno, contadorAnno));
            }


        }
    }


    /**
     * Method promedioPrecio
     *
     * Funcion para calcular un promedio, dividiendo por n
     *
     * @param imp Float del que se quiere calcular el promedio
     * @param n Entero por el que se desea dividir
     * @return Promedio de n / m
     */
    private float promedioPrecio (float imp, int n)
    {
      float promedio = (float) (imp*1.0f)/n;
      return round(promedio, 2);
    }
```

```
/**
 * Method resumenVisitasAtracciones
 *
 * Metodo para genera estadisticas sobre
 * el uso de una atraccion en el periodo
 * de un año
 *
 * @param year año
 * @param atraccion Atraccion
 */
public void resumenVisitasAtracciones(int year, AtraccionIF atraccion)
{

    System.out.println("Atraccion - Tipo " + atraccion.getTipo());
    List<EntradaIF> usuarios = atraccion.getUsuarios();

    analisisPorFechas(usuarios, year);

  }

/**
 * Method analisisPorFechas
 *
 * Funcion base para generar las estadisticas sobre entradas
 * agrupadas por dias, semana, mes y año. Codigo se puede
 * reutilizar, con ligeras modificaciones, para otros
 * tipos de lista
 *
 * @param listaEntradas A parameter
 * @param year Año
 */
private void analisisPorFechas(List<EntradaIF> listaEntradas, int year)
```

```
    {
      int dia = listaEntradas.get(0).getDate().getDayOfMonth();
      int month = listaEntradas.get(0).getDate().getMonthValue();
      int daysInMonth = listaEntradas.get(0).getDate().lengthOfMonth();
      TemporalField                                woy                                =
WeekFields.of(Locale.getDefault()).weekOfWeekBasedYear();
      int semana = listaEntradas.get(0).getDate().get(woy);
      int anno = year;

      int contadorDia = 0;
      int contadorSemana = 0;
      int contadorMes = 0;
      int contadorAnno = 0;

      float promedioSemanal = 0;
      float promedioMes = 0;
      float promedioAnno = 0;

      LocalDate ultimaFecha = listaEntradas.get(listaEntradas.size()-1).getDate();

      System.out.println("Año: " + anno);
      System.out.println("Mes: " + month);
      System.out.println("  Semana: " + semana);

      for (EntradaIF entrada : listaEntradas)
      {
          LocalDate fecha = entrada.getDate();
          if (fecha.getYear() == anno)
          {
              if (fecha.getMonthValue() == month) //Loop para el mes
              {
                  if (fecha.get(woy) == semana)
                  {
                       if (fecha.getDayOfMonth() == dia)
```

```
                    {
                        contadorDia++;
                        contadorSemana++;
                        contadorMes++;
                        contadorAnno++;
                    }
                    else
                    {
                        System.out.println("   " + dia + "/" +
                                           month + "/" +
                                           anno +
                                           "    -    Total    Visitantes:    "    +
contadorDia);
                        contadorDia = 1;
                        dia = fecha.getDayOfMonth();

                        contadorSemana++;
                        contadorMes++;
                        contadorAnno++;
                    }
                }
                else
                {
                    System.out.println("   " + dia + "/" +
                                       month + "/" +
                                       anno +
                                       " - Total Visitantes: " + contadorDia);
                contadorDia = 1;
                dia = fecha.getDayOfMonth();

                System.out.println("    Total Semana: " + contadorSemana);
                promedioSemanal = promedioSemanal(contadorSemana);
                System.out.println("\t\t\t\t          Promedio   Semanal:   "   +
contadorSemana);
                contadorSemana = 1;
```

```
                semana = fecha.get(woy);
                System.out.println("  Semana: " + semana);

                contadorMes++;
                contadorAnno++;
                }
            }
            else
            {
                System.out.println("   " + dia + "/" +
                                   month + "/" +
                                   anno +
                                   " - Total Visitantes: " + contadorDia);
                contadorDia = 1;
                dia = fecha.getDayOfMonth();
                month = fecha.getMonthValue();
                if (fecha.get(woy) != semana)//Casos en los que fin de semana y
fin de mes coinciden
                {
                    System.out.println("    Total Semana: " + contadorSemana);
                    promedioSemanal = promedioSemanal(contadorSemana);
                    System.out.println("\t\t\t\t        Promedio  Semanal:  "  +
contadorSemana);
                    contadorSemana = 1;
                    semana = fecha.get(woy);

                    System.out.println("  Total Mes: " + contadorMes);
                    promedioMes = promedioMensual(contadorMes, daysInMonth);
                    System.out.println("\t\t\t\t        Promedio  Mensual:  "  +
promedioMes);
                    contadorMes = 1;
                    contadorAnno++;

                    System.out.println("Mes: " + month);
                    System.out.println("  Semana: " + semana);
```

```
                }
                else
                {
                    contadorSemana++;

                    System.out.println("  Total Mes: " + contadorMes);
                    promedioMes = promedioMensual(contadorMes, daysInMonth);
                    System.out.println("\t\t\t\t        Promedio  Mensual:  "  +
promedioMes);
                    contadorMes = 1;
                    System.out.println("Mes: " + month);

                    contadorAnno++;
                }
                daysInMonth = fecha.lengthOfMonth();

            }
        }


        if (fecha == ultimaFecha || fecha.getYear() > year)
        {
            dia = fecha.getDayOfMonth();
            month = fecha.getMonthValue();
            daysInMonth = fecha.lengthOfMonth();

            System.out.println("   " + dia + "/" +
                               month + "/" +
                               fecha.getYear() +
                               " - Total Visitantes: " + contadorDia);

            System.out.println("    Total Semana: " + contadorSemana);
            promedioSemanal = promedioSemanal(contadorSemana);
            System.out.println("\t\t\t\t   Promedio Semanal: " + contadorSemana);
```

```
            System.out.println("  Total Mes: " + contadorMes);
            promedioMes = promedioMensual(contadorMes, daysInMonth);
            System.out.println("\t\t\t\t    Promedio Mensual: " + promedioMes);

            System.out.println(" Total Año: " + contadorAnno);
            promedioAnno = promedioAnual(contadorAnno);
            System.out.println("\t\t\t\t    Promedio Anual: " + promedioAnno);
        }
      }
    }

    /**
     * Metodo resumenGastoPersonal
     *
     * Calcula estadisticas sobre los gastos de personal
     * en el parque de atracciones en un año natural,
     * tomando en cuenta las fluctuaciones durante el año
     * dado que ciertas atracciones pueden estar activas o no
     *
     * @param year Año del que se quiere hacer el analisis.
     */
    public void resumenGastoPersonal(int year)
    {

      LocalDate fechaBase = LocalDate.of(year, Month.JANUARY, 1);

      List<Trabajador> trabajadores;
      List<AtraccionIF> atracciones;

      TemporalField                                woy                                =
WeekFields.of(Locale.getDefault()).weekOfWeekBasedYear();
      int semana = fechaBase.get(woy);
      int month = fechaBase.getMonthValue();
```

```
int daysInMonth = fechaBase.lengthOfMonth();
int anno = year;

int contadorSemana = 0;
int contadorMes = 0;
int contadorAnno = 0;

float sumadorDia = 0;
float sumadorSemana = 0;
float sumadorMes = 0;
float sumadorAnno = 0;

float promedioImpSemanal = 0;
float promedioImpMes = 0;
float promedioImpAnno = 0;

System.out.println("Año: " + anno);
System.out.println("Mes: " + month);

for (int i = 0; i < fechaBase.lengthOfYear(); i++)
{
    atracciones = atraccionesActivas.getAtracciones(fechaBase);
    if (atracciones.size() > 0)
    {
          LocalDate fecha = fechaBase;
          if (fecha.getYear() == anno)
          {
              if (fecha.getMonthValue() == month) //Loop para el mes
              {
                  if (fecha.get(woy) == semana)
                  {
                       contadorSemana      =      resolvContador(atracciones,
contadorSemana);
                       sumadorSemana       =       resolvSumador(atracciones,
sumadorSemana);
```

```
                    contadorMes = resolvContador(atracciones,
contadorMes);
                    sumadorMes = resolvSumador(atracciones, sumadorMes);

                    contadorAnno = resolvContador(atracciones,
contadorAnno);
                    sumadorAnno = resolvSumador(atracciones,
sumadorAnno);
                }
                else
                {
                System.out.println("  Semana: " + semana);
                System.out.println("             Total   Semana:   $"   +
round(sumadorSemana, 2));
                System.out.println("\t\t\t\t      Promedio Semanal: $" +
round(promedioPrecio(sumadorSemana, contadorSemana), 2));

                contadorSemana = resolvContador(atracciones, 0);
                sumadorSemana = resolvSumador(atracciones, 0);

                contadorMes = resolvContador(atracciones, contadorMes);
                sumadorMes = resolvSumador(atracciones, sumadorMes);

                contadorAnno = resolvContador(atracciones, contadorAnno);
                sumadorAnno = resolvSumador(atracciones, sumadorAnno);

                semana = fecha.get(woy);
                }
            }
            else
            {
                month = fecha.getMonthValue();
                if (fecha.get(woy) != semana)//Casos en los que fin de
semana y fin de mes coinciden
                {
```

```
                            System.out.println("  Semana: " + semana);
                            System.out.println("           Total   Semana:   $"   +
round(sumadorSemana, 2));
                            System.out.println("\t\t\t\t    Promedio Semanal: $" +
round(promedioPrecio(sumadorSemana, contadorSemana), 2));

                            System.out.println("  Total Mes: $" + sumadorMes);
                            System.out.println("\t\t\t\t    Promedio Mensual: " +
round(promedioPrecio(sumadorMes, contadorMes), 2));

                            contadorSemana = resolvContador(atracciones, 0);
                            sumadorSemana = resolvSumador(atracciones, 0);

                            contadorMes = resolvContador(atracciones, 0);
                            sumadorMes = resolvSumador(atracciones, 0);

                            contadorAnno       =       resolvContador(atracciones,
contadorAnno);
                            sumadorAnno = resolvSumador(atracciones, sumadorAnno);

                            semana = fecha.get(woy);

                            System.out.println("Mes: " + month);
                        }
                        else
                        {

                            System.out.println("  Total Mes: $" + round(sumadorMes,
2));
                            System.out.println("\t\t\t\t    Promedio Mensual: $" +
round(promedioPrecio(sumadorMes, contadorMes), 2));

                            contadorSemana      =      resolvContador(atracciones,
contadorSemana);
```

```
                            sumadorSemana       =       resolvSumador(atracciones,
sumadorSemana);

                            contadorMes = resolvContador(atracciones, 0);
                            sumadorMes = resolvSumador(atracciones, 0);

                            contadorAnno       =       resolvContador(atracciones,
contadorAnno);
                            sumadorAnno = resolvSumador(atracciones, sumadorAnno);

                            System.out.println("Mes: " + month);
                        }
                        daysInMonth = fecha.lengthOfMonth();
                    }
                }
                else
                {

                    System.out.println("    Total Semana: $" + round(sumadorSemana,
2));
                    System.out.println("\t\t\t\t         Promedio  Semanal:  $"  +
round(promedioPrecio(sumadorSemana, contadorSemana), 2));



                    System.out.println("  Total Mes: $" + sumadorMes);
                    System.out.println("\t\t\t\t         Promedio  Mensual:  $"  +
round(promedioPrecio(sumadorMes, contadorMes), 2));

                    System.out.println(" Total Año: $" + sumadorAnno);
                    System.out.println("\t\t\t\t          Promedio  Anual:  $"  +
round(promedioPrecio(sumadorAnno, contadorAnno), 2));



                    contadorSemana = resolvContador(atracciones, 0);
                    sumadorSemana = resolvSumador(atracciones, 0);
```

```
                    contadorMes = resolvContador(atracciones, 0);
                    sumadorMes = resolvSumador(atracciones, 0);

                    contadorAnno = resolvContador(atracciones, 0);
                    sumadorAnno = resolvSumador(atracciones, 0);

                    semana = fecha.get(woy);
                    month = fecha.getMonthValue();
                    anno = fecha.getYear();

                    System.out.println("Año: " + anno);
                    System.out.println("Mes: " + month);
                    System.out.println("  Semana: " + semana);

                }

                if (i == fechaBase.lengthOfYear() - 1)
                {
                    month = fecha.getMonthValue();
                    daysInMonth = fecha.lengthOfMonth();

                    System.out.println("    Total Semana: $" + round(sumadorSemana,
2));
                    System.out.println("\t\t\t\t         Promedio  Semanal:  "  +
round(promedioPrecio(sumadorSemana, contadorSemana), 2));

                    System.out.println("  Total Mes: $" + sumadorMes);
                    System.out.println("\t\t\t\t         Promedio  Mensual:  "  +
round(promedioPrecio(sumadorMes, contadorMes), 2));

                    System.out.println(" Total Año: $" + sumadorAnno);
                    System.out.println("\t\t\t\t          Promedio  Anual:  $"  +
round(promedioPrecio(sumadorAnno, contadorAnno), 2));
                }
                else
```

```
            {
                fechaBase = fechaBase.plusDays(1);
            }
      }
      else
      {
        fechaBase = fechaBase.plusDays(1);
      }
  }
}

/**
 * Method resolvContador
 *
 * Funcion de ayuda para las estadisticas de gasto personal,
 * que se encarga de actualizar un contador de trabajadores
 * a partir de la lista de atracciones suministrada
 *
 * @param atracciones Lista de atracciones a analizar
 * @param n Valor actual del contador (entero)
 * @return Valor actualizado del contador (entero)
 */
private int resolvContador(List<AtraccionIF> atracciones, int n)
{
  List<Trabajador> trabajadores;
  int contador = n;
  for (AtraccionIF atraccion : atracciones)
  {
    trabajadores = atraccion.getTrabajadores();
    for (Trabajador trabajador : trabajadores)
    {
      contador++;
    }
  }
```

```
      for (Trabajador trabajador : ListaTrabajadores)
      {
        if    (trabajador.getTipo()    ==    TiposTrabajadores.REL_PUBLICAS    ||
trabajador.getTipo() == TiposTrabajadores.ATENCION_CL)
        {
          contador++;
        }
      }

      return contador;
    }

    /**
     * Method resolvSumador
     *
     * Funcion de ayuda para las estadisticas de gasto personal,
     * que se encarga de actualizar un sumador de salario de los trabajadores
     * a partir de la lista de atracciones suministrada
     *
     * @param atracciones Lista de atracciones a analizar
     * @param n Valor actual del sumador (float)
     * @return Valor actualizado del sumador (float)
     */
    private float resolvSumador(List<AtraccionIF> atracciones, float n)
    {
      List<Trabajador> trabajadores;
      float sumador = n;
      for (AtraccionIF atraccion : atracciones)
      {
        trabajadores = atraccion.getTrabajadores();
        for (Trabajador trabajador : trabajadores)
        {
          sumador += trabajador.getSueldo();
        }
```

```
      }
      for (Trabajador trabajador : ListaTrabajadores)
      {
        if    (trabajador.getTipo()    ==    TiposTrabajadores.REL_PUBLICAS    ||
trabajador.getTipo() == TiposTrabajadores.ATENCION_CL)
        {
          sumador++;
        }
      }
      return sumador;
    }

    /**
     * Method round
     *
     * Metodo para redondear valores float a dos decimales
     *
     * @param d Valor a redondear (float)
     * @param decimalPlace Numero de decimales (entero)
     * @return Valor redondeado (float)
     */
    private float round(float d, int decimalPlace)
    {
        BigDecimal n = new BigDecimal(Float.toString(d));
        n = n.setScale(decimalPlace, BigDecimal.ROUND_HALF_UP);
        return n.floatValue();
    }
}
```

GeneradorContenido (versión programa completo)

```
/**
 * Clase para generar contenido automatico
 * para el parque
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */
public class GeneradorContenido
{
    public static void generadorContenido(ParqueManager manager)
    {
        int nTrabajadores;
        //Crear atracciones
        for (int i = 0; i < 4; i++)
        {
            manager.addAtraccion("A");
        }

        for (int i = 0; i < 6; i++)
        {
            manager.addAtraccion("B");
        }

        for (int i = 0; i < 4; i++)
        {
            manager.addAtraccion("C");
        }

        for (int i = 0; i < 3; i++)
```

```
        {
            manager.addAtraccion("D");
        }

        for (int i = 0; i < 7; i++)
        {
            manager.addAtraccion("E");
        }

        nTrabajadores = manager.getNumTrabajadores();

        int nAteCl = (int)(nTrabajadores * 0.3);

        for (int i = 0; i < nAteCl; i++)
        {
            manager.addTrabajador(TiposTrabajadores.ATENCION_CL);
        }

        int nRelPubl = (int)(nTrabajadores * 0.1);

        for (int i = 0; i < nRelPubl; i++)
        {
            manager.addTrabajador(TiposTrabajadores.REL_PUBLICAS);
        }


    }
}
```

GeneradorVisitantes

```
import java.io.*;
import static java.lang.System.*;
import java.time.*;

public class GeneradorVisitantes
{


    public static void generarVisitantes (ParqueManager manager)
    {
    int tipo = 0;

    int edad = 0;
    boolean familia = false;
    boolean vip = false;
    Month month;
    LocalDate fecha;
    String[] tipos = { "niño", "senior", "carnet joven", "discapacitado",
"estudiante", "veterano"};

    try
    {
        BufferedReader br = new BufferedReader(new
FileReader("Visitantes.txt"));
        String line = null;

        while ((line = br.readLine()) != null) {
            String tmp[] = line.split(";");
            tipo = Integer.parseInt(tmp[0]);

            edad = Integer.parseInt(tmp[1]);
            if (Integer.parseInt(tmp[2]) == 1)
            {
                vip = true;
            }
            if (Integer.parseInt(tmp[3]) == 1)
            {
                familia = true;
            }
             switch (Integer.parseInt(tmp[5]))
             {
                 case 1:
                    month = Month.JANUARY;
                    break;
                 case 2:
                    month = Month.FEBRUARY;
                    break;
```

```
                case 3:
                   month = Month.MARCH;
                   break;
                case 4:
                   month = Month.APRIL;
                   break;
                case 5:
                   month = Month.MAY;
                   break;
                case 6:
                   month = Month.JUNE;
                   break;
                case 7:
                   month = Month.JULY;
                   break;
                case 8:
                   month = Month.AUGUST;
                   break;
                case 9:
                   month = Month.SEPTEMBER;
                   break;
                case 10:
                   month = Month.OCTOBER;
                   break;
                case 11:
                   month = Month.NOVEMBER;
                   break;
                case 12:
                   month = Month.DECEMBER;
                   break;
                default:
                   month = Month.JANUARY;
             }
            fecha = LocalDate.of(2019, month, Integer.parseInt(tmp[4]));
            //System.out.println("Visitante: " + tipos[tipo]);

            if (tipo == 0)
            {
                manager.addEntradaNiño(fecha, edad, vip, familia,
"ninguno");
            }
            else
            {
                manager.addEntrada(fecha, edad, vip, familia,
tipos[tipo]);
            }

        }
    }
```

```
    catch (IOException e)
    {
        e.printStackTrace();
    }
}
}
```

parque (metodo main)

```
/**
 * Clase con el metodo main del sistema.
 *
 * @author (Samuel Alarco)
 * @version (v1.0)
 */

public class parque
{

    public static void main (String[] args)
    {
        ParqueManager manager = new ParqueManager();
        MenuInterface menu = new MenuInterface(manager);
        menu.menu1();

    }

}
```

MenuInterface

```
/**
 * Clase que contiene toda la funcionalidad de
 * los menus que controlan el programa.
 *
 * @author (your name)
 * @version (a version number or a date)
 */

import java.util.Scanner;
import java.time.*;
import java.time.format.*;
import java.util.HashMap;
import java.util.List;
import java.util.LinkedList;


public class MenuInterface
{
  Scanner input = new Scanner(System.in);
  ParqueManager manager;
  AnalizadorEstadisticas analizador;
```

```
public MenuInterface(ParqueManager manager)
{
    this.manager = manager;
    analizador = manager.analisisEstadistico();
}

/**
 * Method menu1
 *
 * Menu Principal del programa
 *
 */
public void menu1()
 {
     imprLineaBl();
     System.out.println("-- Menu Principal --");
     System.out.println(
     "Seleccione una opcion: \n" +
     "  1) Menu Entradas\n" +
     "  2) Menu Atracciones\n" +
     "  3) Menu Trabajadores \n" +
     "  4) Menu Estadisticas\n" +
     "  5) Salir\n "
     );

     int seleccion = this.input.nextInt();
     input.nextLine();

     switch (seleccion) {
     case 1:
       System.out.println("=========================");
       this.menuEntradas();
       break;
     case 2:
       System.out.println("=========================");
       this.menuAtracciones();
       break;
     case 3:
       System.out.println("=========================");
       menuTrabajadores();
       break;
     case 4:
       System.out.println("=========================");
       menuEstadisticas();
       break;
     case 5:
         System.exit(1);
     default:
       System.out.println("Entrada Invalida.");
```

```
      break;
    }
}

/////////////////MENUS ENTRADA

/**
 * Method menuEntradas
 *
 * Menu principal para toda la funcionalidad
 * de las entradas
 *
 */
private void menuEntradas()
{
    imprLineaBl();
    System.out.println("-- Menu Entradas --");
    System.out.println(
    "Seleccione una opcion: \n" +
    "  1) Nueva Entrada Individual\n" +
    "  2) Importar Entradas desde Archivo\n" +
    "  3) Resumen Entradas \n" +
    "  4) Salir\n "
    );

    int seleccion = this.input.nextInt();
    input.nextLine();

    switch (seleccion) {
    case 1:
      System.out.println("=========================");
      this.menuNuevaEntrada();
      break;
    case 2:
      GeneradorVisitantes.generarVisitantes(manager);
      this.menuEntradas();
      break;
    case 3:
      resumenVisitantes();
      break;
    case 4:
        this.menu1();
    default:
      System.out.println("Entrada Invalida.");
      break;
    }
}


/**
```

```
 * Method menuNuevaEntrada
 *
 * Menu para crear nueva entrada
 *
 */
private void menuNuevaEntrada()
{
    imprLineaBl();
    System.out.println("-- Nueva Entrada --");
    System.out.println(
    "Seleccione una opcion: \n" +
    "  1) Nueva Entrada Adulto\n" +
    "  2) Nueva Entrada Niño\n" +
    "  3) Salir\n "
    );

    int seleccion = this.input.nextInt();
    input.nextLine();

    switch (seleccion) {
    case 1:
      System.out.println("=========================");
      this.menuNuevaEntradaGeneral(false);
      break;
    case 2:
      this.menuNuevaEntradaGeneral(true);
      break;
    case 3:
      this.menu1();
      break;
    default:
      System.out.println("Entrada Invalida.");
      break;
    }
}

/**
 * Method menuNuevaEntradaGeneral
 *
 * Menu para configurar y crear una nueva entrada
 * General o de niño
 *
 * @param nino True si entrada es de niño
 */
private void menuNuevaEntradaGeneral(boolean nino)
{
    try {
    if(!nino) {System.out.println("-- Nueva Entrada General --");}
    else {System.out.println("-- Nueva Entrada Niño --");}
```

```
        System.out.print("Fecha de compra (dd/MM/uuuu):");
        String fechaIn = input.nextLine();
        LocalDate fecha;
        DateTimeFormatter formato = DateTimeFormatter.ofPattern (
"dd/MM/uuuu" );
        fecha = LocalDate.parse ( fechaIn , formato );
        //System.out.println ( "ld: " + fecha );

        System.out.print("Edad: ");
        int edad = input.nextInt();
        if (!nino)
        {
            while (edad < 13)
            {
                System.out.println("Edad invalida");
                System.out.print("Edad (debe ser mayor de 12 años):");
                edad = input.nextInt();
            }
        }
        else
        {
            while (edad < 0 || edad > 12)
            {
                System.out.println("Edad invalida");
                System.out.print("Edad (debe ser menor de 13 años):");
                edad = input.nextInt();
            }
        }
        input.nextLine();

        System.out.print("Entrada VIP? (Y/N):");
        String VIPStr = input.nextLine();
        boolean vip;
        while (!VIPStr.equals("Y") && !VIPStr.equals("N"))
        {
            System.out.println("Decision invalida");
            System.out.print("Entrada VIP? (Y/N):");
            VIPStr = input.nextLine();
        }
        if (VIPStr.equals("Y")) {vip = true;}
        else {vip = false;}

        System.out.print("Entrada Familiar? (Y/N):");
        String familiarStr = input.nextLine();
        boolean fam;
        while (!familiarStr.equals("Y") && !familiarStr.equals("N"))
        {
            System.out.println("Decision invalida");
            System.out.print("Entrada Familiar? (Y/N)::");
            familiarStr = input.nextLine();
```

```
}
if (familiarStr.equals("Y")) {fam = true;}
else {fam = false;}
imprLineaBl();

System.out.print("Descuentos Disponibles:\n");
HashMap<String, Float> descuentos = manager.obtenerDescuentos();
for (String key : descuentos.keySet())
{
    System.out.println(key + ": " + descuentos.get(key) + "%");
}
imprLineaBl();
System.out.print("Desea Añadir algun Descuento? (Y/N):");
String addDescuento = input.nextLine();
while (!addDescuento.equals("Y") && !addDescuento.equals("N"))
{
    System.out.println("Decision invalida");
    imprLineaBl();
    System.out.print("Desea Añadir algun Descuento? (Y/N):");
    addDescuento = input.nextLine();
}
String descuentoStr = "";
if (addDescuento.equals("N"))
{
    descuentoStr = "ninguno";
}
else
{
     while (addDescuento.equals("Y"))
     {
        System.out.print("Introducir nombre del descuento: ");
        String descuentoIn = input.nextLine();
        if (descuentos.containsKey(descuentoIn))
        {
            descuentoIn += ";";
            descuentoStr += descuentoIn;
            System.out.println(descuentoStr);
        }
        else
        {
            System.out.println("Descuento no existe");
        }
        imprLineaBl();
        System.out.print("Desea Añadir algun Descuento? (Y/N):");
        addDescuento = input.nextLine();
     }
}


//Imprimir Resumen
```

```
        System.out.println("Resumen de la Entrada:");
        System.out.println("\tFecha: " + fecha);
        System.out.println("\tEdad: " + edad);
        if (vip)
        {System.out.println("\tVIP: SI");}
        else
        {System.out.println("\tVIP: NO");}

        if (fam)
        {System.out.println("\tFamiliar: SI");}
        else
        {System.out.println("\tFamiliar: NO");}

        System.out.print("\t");
        String tmp[] = descuentoStr.split(";");
        for (int i = 0; i < tmp.length; i++)
        {
            System.out.print(tmp[i] + " - ");
        }
        imprLineaBl();

        System.out.println("Confirmar Entrada? (Y/N)");
        String confirmar = input.nextLine();
        while (!confirmar.equals("Y") && !confirmar.equals("N"))
        {
            System.out.println("Decision invalida");
            imprLineaBl();
            System.out.print("Confirmar Entrada? (Y/N):");
            addDescuento = input.nextLine();
        }
        if (confirmar.equals("Y"))
        {
            imprLineaBl();
            if (!nino) {manager.addEntrada(fecha, edad, vip, fam,
descuentoStr);}
            else {manager.addEntradaNiño(fecha, edad, vip, fam,
descuentoStr);}
            menuNuevaEntrada();
        }
        else
        {
            System.out.println("Entrada Cancelada");
            menuNuevaEntrada();
        }

        }
        catch ( DateTimeParseException e ) {
            System.out.println ( "Fecha Invalida");
            this.menuNuevaEntrada();
        }
```

```
}

/**
 * Method resumenVisitantes
 *
 * Opcion para mostrar resumen de visitantes
 * del parque
 *
 */
private void resumenVisitantes()
{
    imprLineaBl();
    System.out.println("-- Resumen Visitantes --");
    analizador.resumenVisitantesTipo();
    menuEntradas();
}

////////////// MENUS ATRACCIONES

/**
 * Method menuAtracciones
 *
 * Menu principal para la funcionalidad de
 * de Atracciones
 *
 */
private void menuAtracciones()
{
  imprLineaBl();
  System.out.println("-- Menu Atracciones --");
  System.out.println(
  "Seleccione una opcion: \n" +
  "  1) Nueva Atraccion\n" +
  "  2) Generar Atracciones\n" +
  "  3) Activar/Desactivar Atracciones \n" +
  "  4) Generar Uso Aleatorio de Atracciones\n" +
  "  5) Resumen Atracciones\n" +
  "  6) Salir\n "
  );

  int seleccion = this.input.nextInt();
  input.nextLine();

  switch (seleccion) {
  case 1:
    imprLineaBl();
    this.menuNuevaAtraccion();
    break;
  case 2:
    generarAtracciones();
```

```
        break;
      case 3:
        menuAtraccionesActivas();
        break;
      case 4:
        usoAleatorioAtracciones();
        break;
      case 5:
        resumenAtracciones();
        break;
      case 6:
          menu1();
      default:
        System.out.println("Entrada Invalida.");
        break;
      }

    }

    /**
     * Method menuNuevaAtraccion
     *
     * Menu para crear una nueva atraccion
     *
     */
    private void menuNuevaAtraccion()
    {
      imprLineaBl();
      System.out.print("Introducir Tipo de Atraccion (A, B, C, D, E): ");
      String tipo = input.nextLine();
      while (!tipo.equals("A") &&
             !tipo.equals("B") &&
             !tipo.equals("C") &&
             !tipo.equals("D") &&
             !tipo.equals("E")
             )
      {
        System.out.println("Tipo Invalido");
        System.out.print("Introducir Tipo de Atraccion (A, B, C, D, E):
");
        tipo = input.nextLine();
      }
      manager.addAtraccion(tipo);
      System.out.println("Atraccion Añadida con exito");
      menuAtracciones();

    }

    /**
     * Method menuAtraccionesActivas
```

```
 *
 * Menu para administrar atracciones activas
 *
 */
private void menuAtraccionesActivas()
{
  imprLineaBl();
  System.out.println("-- Menu Atracciones --");
  System.out.println(
  "Seleccione una opcion: \n" +
  "  1) Nuevo Periodo de Atracciones Activas\n" +
  "  2) Generar Atracciones Activas\n" +
  "  3) Resumen Atracciones Activas\n" +
  "  4) Salir"
  );

  int seleccion = this.input.nextInt();
  input.nextLine();

  switch (seleccion) {
  case 1:
    imprLineaBl();
    this.menuNuevaAtraccionActiva();
    break;
  case 2:
    generarAtraccionesActivas();
    break;
  case 3:
    resumenAtraccionesActivas();
    menuAtracciones();
    break;
  case 4:
    menuAtracciones();
    break;

  default:
    System.out.println("Entrada Invalida.");
    break;
  }

}


/**
 * Method generarAtracciones
 *
 * Menu para generar atracciones del parque
 * segun el enunciado
 *
 */
```

```
    private void generarAtracciones()
    {
      GeneradorContenido.generadorContenido(manager);
      imprLineaBl();
      System.out.println("-- Lista de Atracciones --");
      analizador.resumenAtracciones();
      menuAtracciones();
    }

    /**
     * Method resumenAtracciones
     *
     * Opcion para mostrar resumen de las
     * atracciones presentes en el parque
     *
     */
    private void resumenAtracciones()
    {
      imprLineaBl();
      System.out.println("-- Lista de Atracciones --");
      analizador.resumenAtracciones();
      menuAtracciones();
    }

    /**
     * Method menuNuevaAtraccionActiva
     *
     * Menu para crear nuevos periodos de atracciones
     * activas
     *
     */
    private void menuNuevaAtraccionActiva()
    {
      imprLineaBl();
      System.out.println("-- Nuevo Periodo de Atracciones Activas --");
      imprLineaBl();
      System.out.println("Periodos Actuales:");
      resumenAtraccionesActivas();
      imprLineaBl();
      System.out.println("Creacion de Nuevo Periodo:");
      System.out.println("* Crea un nuevo periodo. Primero especifique las
fechas del periodo\n" +
                         "* y despues escoja las atracciones que deben
estar activas durante ese periodo\n" +
                         "* Para modificar un periodo existente, entre las
mismas fechas\n" +
                         "* y cree una lista nueva. La lista nueva
reemplazara la vieja");

      System.out.print("Fecha Inicial: ");
```

```
      String fechaInic = input.nextLine();
      LocalDate fechaInicial;
      DateTimeFormatter formato = DateTimeFormatter.ofPattern (
"dd/MM/uuuu" );
      fechaInicial = LocalDate.parse ( fechaInic , formato );

      System.out.print("Fecha Final: ");
      String fechaFin = input.nextLine();
      LocalDate fechaFinal;
      fechaFinal = LocalDate.parse ( fechaFin , formato );

      PeriodoTemporada periodo = new PeriodoTemporada(fechaInicial,
fechaFinal);

      imprLineaBl();

      List<AtraccionIF> atraccionesDisponibles = manager.getAtracciones();
      List<AtraccionIF> atraccionesSeleccionadas = new
LinkedList<AtraccionIF>();

      System.out.println("Atracciones Disponibles: ");
      int counter = 0;
      for (AtraccionIF atraccion : atraccionesDisponibles)
      {
        System.out.println("Atraccion #" + counter + " tipo " +
atraccion.getTipo());
        counter++;
      }

      imprLineaBl();

      String confirmar;
      int atraccionIndex = 0;
      do
      {
        System.out.print("Desea Añadir una Atraccion? (Y/N): ");
        confirmar = input.nextLine();
      }
      while (!confirmar.equals("Y") && !confirmar.equals("N"));

      while(confirmar.equals("Y"))
      {
        System.out.print("Seleccione una Atraccion para añadir\n" +
                         "a la lista de atracciones activas en el periodo
de tiempo especificado (numero): ");
        if (atraccionIndex > counter)
        {
          System.out.println("Entrada No Valida");
        }
        else
```

```
        {
          atraccionIndex = input.nextInt();

atraccionesSeleccionadas.add(atraccionesDisponibles.get(atraccionIndex));
          System.out.println("Atracciones Seleccionadas:");
          for (AtraccionIF atraccion : atraccionesSeleccionadas)
          {
            System.out.print("Atraccion " +
atraccionesSeleccionadas.indexOf(atraccion) + " tipo: " +
atraccion.getTipo() + " - ");
          }

          do
          {
            System.out.println("Desea Añadir una Atraccion? (Y/N): ");
            confirmar = input.nextLine();
          }
          while (!confirmar.equals("Y") && !confirmar.equals("N"));
        }

      }
      manager.addAtraccionesFuncionando(periodo,
atraccionesSeleccionadas);

      resumenAtraccionesActivas();
      menuAtracciones();

    }

    /**
     * Method resumenAtraccionesActivas
     *
     * Opcion para mostrar resumen de los
     * periodos de atracciones activas
     *
     */
    private void resumenAtraccionesActivas()
    {
      analizador.resumenAtraccionesActivas();
    }

    /**
     * Method generarAtraccionesActivas
     *
     * Menu para generar periodos de atracciones
     * activas automaticamente
     *
     */
    private void generarAtraccionesActivas()
    {
```

```
  manager.setContenidoAtraccionesFuncionando();
  menuAtraccionesActivas();
}

/**
 * Method usoAleatorioAtracciones
 *
 * Menu para generar uso aleatorio de
 * las atracciones
 *
 */
private void usoAleatorioAtracciones()
{
  manager.randomUsarAtracciones();
  imprLineaBl();
  System.out.println("Uso aleatorio de las atracciones generado");
  menuAtracciones();
}

///////// MENUS TRABAJADORES

/**
 * Method menuTrabajadores
 *
 * Menu principal para la funcionalidad
 * de los trabajadores
 *
 */
private void menuTrabajadores()
{
  imprLineaBl();
  System.out.println("-- Menu Atracciones --");
  System.out.println(
  "Seleccione una opcion: \n" +
  "  1) Resumen de los Trabajadores\n" +
  "  2) Salir\n "
  );

  int seleccion = this.input.nextInt();
  input.nextLine();

  switch (seleccion) {
  case 1:
    imprLineaBl();
    resumenTrabajadores();;
    break;
  case 2:
    menu1();
    break;
  default:
```

```
        System.out.println("Entrada Invalida.");
        break;
      }
    }


    /**
     * Method resumenTrabajadores
     *
     * Opcion para mostrar resumen de los
     * trabajadores del parque de atracciones
     *
     */
    private void resumenTrabajadores()
    {
      System.out.println("-- Resumen Trabajadores --");
      System.out.println("Los trabajadores se generan automaticamente\n" +
                         "segun las proporciones indicadas cuando se
crean\n" +
                         "nuevas atracciones:");
      analizador.trabajadoresPorAtraccion();
      menuTrabajadores();
    }

    ////////// MENU Estadisticas

    /**
     * Method menuEstadisticas
     *
     * Menu principal de la funcionalidad
     * de estadisticas del sistema
     *
     */
    private void menuEstadisticas()
    {
      imprLineaBl();
      System.out.println("-- Menu Estadisticas --");
      System.out.println(
      "Seleccione una opcion: \n" +
      "  1) Informacion Visitantes del Parque\n" +
      "  2) Informacion Precio Entradas\n" +
      "  3) Informacion Uso de Atracciones\n" +
      "  4) Informacion Gastos del Personal Diario\n" +
      "  5) Salir"
      );

      int seleccion = this.input.nextInt();
      input.nextLine();

      switch (seleccion) {
```

```
  case 1:
    imprLineaBl();
    estadisticasVisitantes();
    break;
  case 2:
    imprLineaBl();
    estadisticasPrecios();
    break;
  case 3:
    imprLineaBl();
    estadisticasUsoAtracciones();
    break;
  case 4:
    imprLineaBl();
    estadisticasGastos();
    break;
  case 5:
    imprLineaBl();
    menu1();
  default:
    System.out.println("Entrada Invalida.");
    break;
  }
}

/**
 * Method estadisticasVisitantes
 *
 * Opcion para generar estadisticas sobre los
 * visitantes que visitan el parque
 *
 */
private void estadisticasVisitantes()
{
  System.out.println("Especificar Año: ");
  int anno = input.nextInt();
  imprLineaBl();
  analizador.resumenVisitantes(anno);
  menuEstadisticas();
}

/**
 * Method estadisticasPrecios
 *
 * Opcion para generar estadisticas
 * sobre los precios de las entradas
 *
 */
private void estadisticasPrecios()
{
```

```
      System.out.print("Especificar Año: ");
      int anno = input.nextInt();
      imprLineaBl();
      analizador.resumenPrecios(anno);
      menuEstadisticas();
    }


    /**
     * Method estadisticasUsoAtracciones
     *
     * Opcion para generar estadisticas sobre
     * el uso de las atracciones
     *
     */
    private void estadisticasUsoAtracciones()
    {
      System.out.print("Especificar Año: ");
      int anno = input.nextInt();
      List<AtraccionIF> atraccionesDisponibles = manager.getAtracciones();
      System.out.println("Atracciones Disponibles: ");
      int counter = 0;
      for (AtraccionIF atraccion : atraccionesDisponibles)
      {
        System.out.println("Atraccion #" + counter + " tipo " +
atraccion.getTipo());
        counter++;
      }

      int atraccionIndex = 0;
      System.out.print("Seleccione una Atraccion (numero): ");
      atraccionIndex = input.nextInt();

      while (atraccionIndex > counter)
      {
        System.out.println("Entrada No Valida");
        System.out.print("Seleccione una Atraccion (numero): ");
        atraccionIndex = input.nextInt();
      }

      imprLineaBl();
      analizador.resumenVisitasAtracciones(anno,
atraccionesDisponibles.get(atraccionIndex));
      menuEstadisticas();
    }

    /**
     * Method estadisticasGastos
     *
     * Opcion para generar estadisticas sobre los gastos
```

```
     * diarios del personal del parque
     *
     */
    private void estadisticasGastos()
    {
      System.out.print("Especificar Año: ");
      int anno = input.nextInt();
      imprLineaBl();
      analizador.resumenGastoPersonal(anno);
      menuEstadisticas();
    }

    /////////// OTROS

    private void imprLineaBl()
    {
      System.out.println();
    }

}
```