TPA Tercera Entrega
Equipo 4

UTN.BA
DPTO. INGENIERÍA EN SISTEMAS DE INFORMACIÓN
CÁTEDRA DISEÑO DE SISTEMAS

Alcance

La entrega comprende:

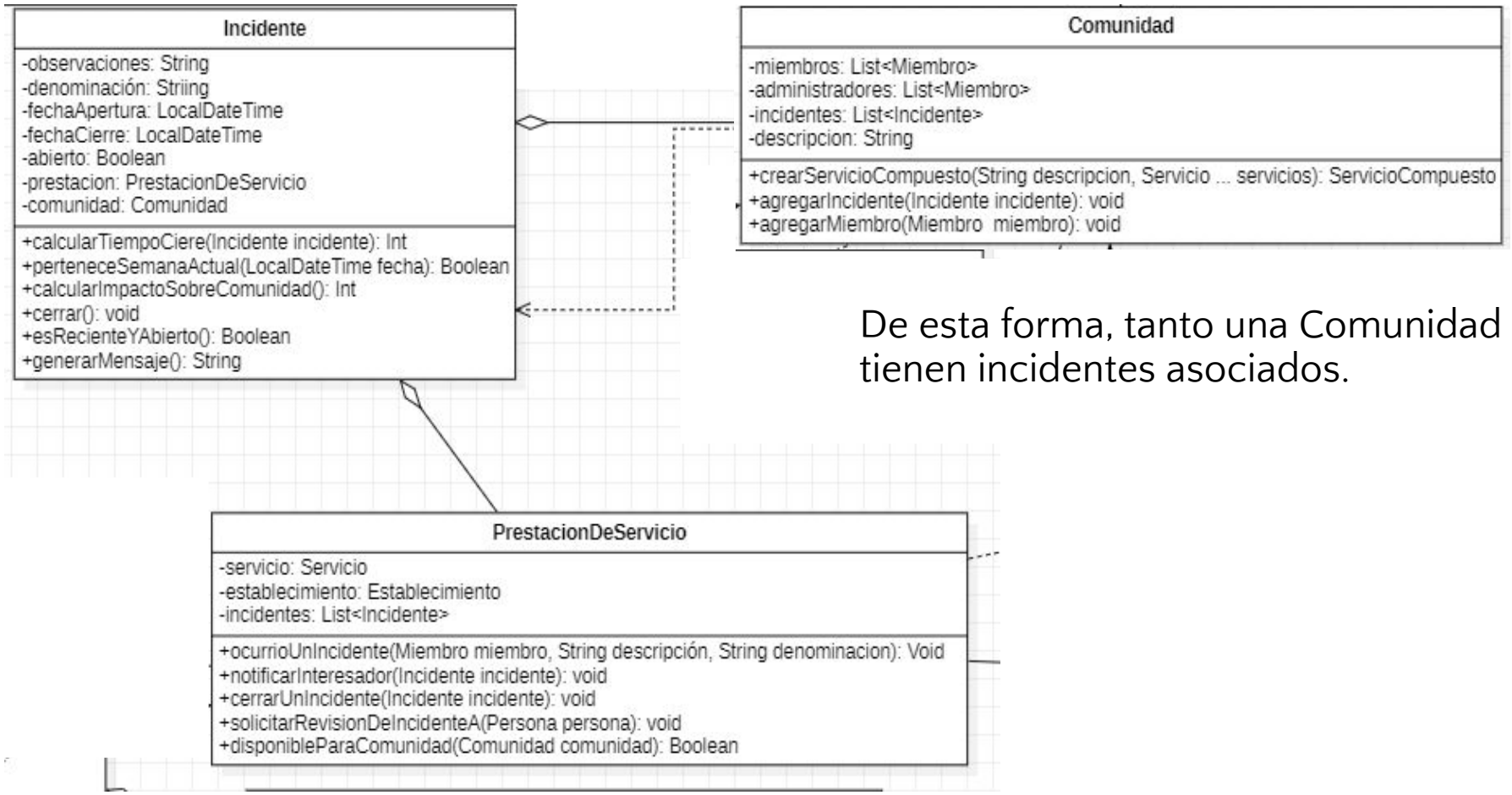
- Apertura de Incidentes sobre Prestaciones de Servicios
- Cierre de Incidentes
- Envío de notificaciones
    - Por medios:
        - Email
        - WhatsApp
    - Por eventos de:
        - Apertura de incidentes
        - Cierre de incidentes
        - Sugerencia de revisión de incidentes
    - Según forma:
        - Cuando suceden (instantáneas)
        - Sin apuros (diferidas en el tiempo)
- Ranking de Incidentes según los tres criterios
- Informes para entidades prestadoras/organismos de control de Rankings
- Miembros Afectados/Observadores

Incidentes

Diseño del Incidente; apertura de incidentes; cierre de incidentes.

Incidentes – Diseño general

Los incidentes asociados a Comunidades y prestaciones de servicio, tienen el siguiente diseño:



De esta forma, tanto una Comunidad como una Entidad tienen incidentes asociados.

Incidentes – Apertura de incidentes

Los incidentes se abren cuando ocurren en una prestación de servicio, se guardan también en la comunidad y se notifica a los interesados.

```
// PRESTACION DE SERVICIO
public void ocurrioUnIncidente(Miembro miembro, String descripcion, String denominacion) {
  Comunidad comunidadMiembro = miembro.getComunidad();
  Incidente incidente = new Incidente(descripcion, denominacion, comunidadMiembro,  prestacion: this);
  incidentes.add(incidente);
  comunidadMiembro.agregarIncidente(incidente);
  notificarInteresados(incidente);
}
```

Incidentes – Cierre de incidentes

Tomamos la decisión de que los incidentes se cierren desde la prestación.

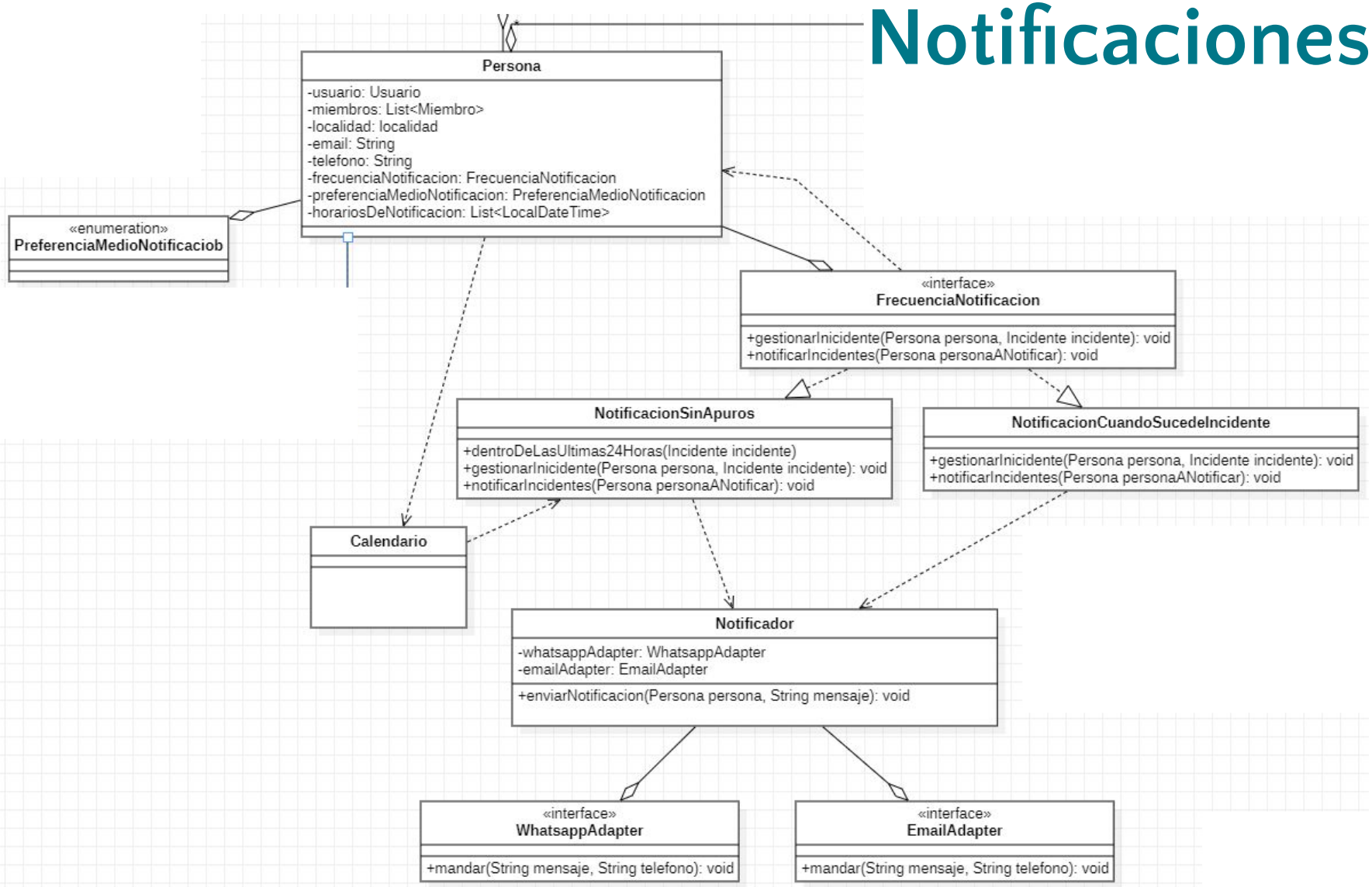
```
// PRESTACION DE SERVICIO
public void cerrarUnIncidente(Incidente incidente) {
  incidente.cerrar();
  notificarInteresados(incidente);
}
```

```
// INCIDENTE
public void cerrar() {
  setFechaCierre(LocalDateTime.now());
  setAbierto(false);
}
```

Notificaciones

Diseño del Módulo de Notificaciones; notificaciones por los distintos medios; envío de notificaciones según eventos; envío de notificaciones en el momento y de forma diferida en el tiempo.

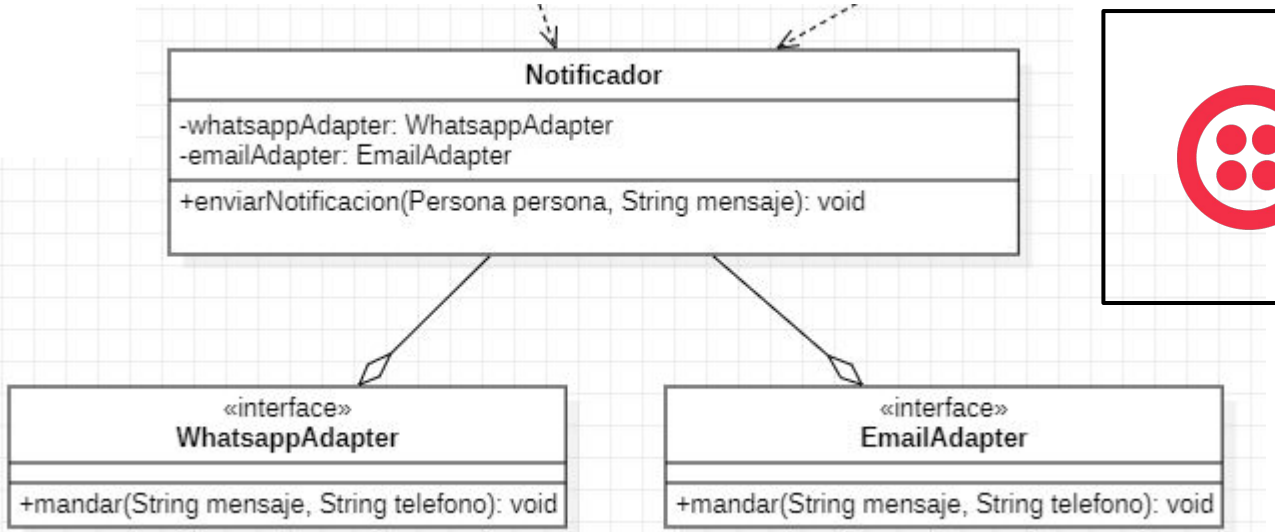
Notificaciones – Diseño general



Decidimos implementar una clase notificador (singleton) para manejar el envío de notificaciones y manejarlo desde la frecuencia de la notificación.

Notificaciones – Medios de Notificaciones

Para esta funcionalidad se utilizó un enumerado para saber que tipo de medio de notificación tiene definido la persona. Se utilizaron las APi´s de Twilio y Jakarta para implementarlo.



twilio

JAKARTA EE

Notificaciones – Eventos que generan notificaciones

Comentar las decisiones de diseño más relevantes..

Puede agregarse diagrama de clases (parcial) para mostrar este diseño.

Notificaciones – Envío sincrónico/asincrónico

Envío sincrónico:

```java
@Override
public void gestionarInicidente(Persona persona, Incidente incidente) {
  String mensaje = incidente.generarMensaje();
  Notificador.obtenerInstancia().enviarNotificacion(persona, mensaje); // porque es un Singleton
}
```



```java
public void enviarNotificacion(Persona persona, String mensaje) {
  switch (persona.getPreferenciaMedioNotificacion()){
    case WHATSAPP :
      whatsappAdapter.mandar(mensaje, persona.getTelefono());
      break;

    case EMAIL:
      emailAdapter.mandar(mensaje, persona.getEmail());
      break;

    default:
      //TODO excepcion
      break;
  }
```

Twilio:

```java
public void mandar (String mensaje, String telefono) {
  Twilio.init(ACCOUNT_SID, AUTH_TOKEN);
  Message message = Message.creator(
      new com.twilio.type.PhoneNumber("whatsapp:+5491134099892"),
      new com.twilio.type.PhoneNumber("whatsapp:+14155238886"),
      mensaje).create();

  System.out.println(message.getSid());
}
```

Jakarta:

```java
try {
  MimeMessage message = new MimeMessage(session);
  message.setFrom(new InternetAddress(from));
  message.setRecipients(Message.RecipientType.TO, InternetAddress.parse(email));
  message.setSubject(subject);
  message.setText(mensaje);

  Transport.send(message);
  System.out.println("Se mando el email exitosamente.");
} catch (MessagingException e) {
  e.printStackTrace();
}
```

Notificaciones – Envío asincrónico

Calendario:

```java
public class Calendario {
  public Calendario(Persona persona) throws SchedulerException {
    Scheduler scheduler = StdSchedulerFactory.getDefaultScheduler();
    scheduler.start();

    int contadorHorarios = 0;
    for (LocalDateTime horario : persona.getHorariosDeNotificaciones()) {
      String nombreGrupo = persona.getEmail() + contadorHorarios;
      JobDetail job = JobBuilder.newJob(NotificacionSinApuros.class)
          .withIdentity( name: "miNotificacion", nombreGrupo)
          .build();

      JobDataMap jobDataMap = job.getJobDataMap();
      jobDataMap.put("persona", persona);

      Trigger trigger = TriggerBuilder.newTrigger() TriggerBuilder<Trigger>
          .withIdentity( name: "myTrigger", nombreGrupo)
          .withSchedule(CronScheduleBuilder.dailyAtHourAndMinute(horario.getHour(), horario.getMinute()))
          .build();

      contadorHorarios = contadorHorarios + 1 ;

      // Schedule the job with the trigger
      scheduler.scheduleJob(job, trigger);
    }
  }
```

Notificador Sin Apuros:

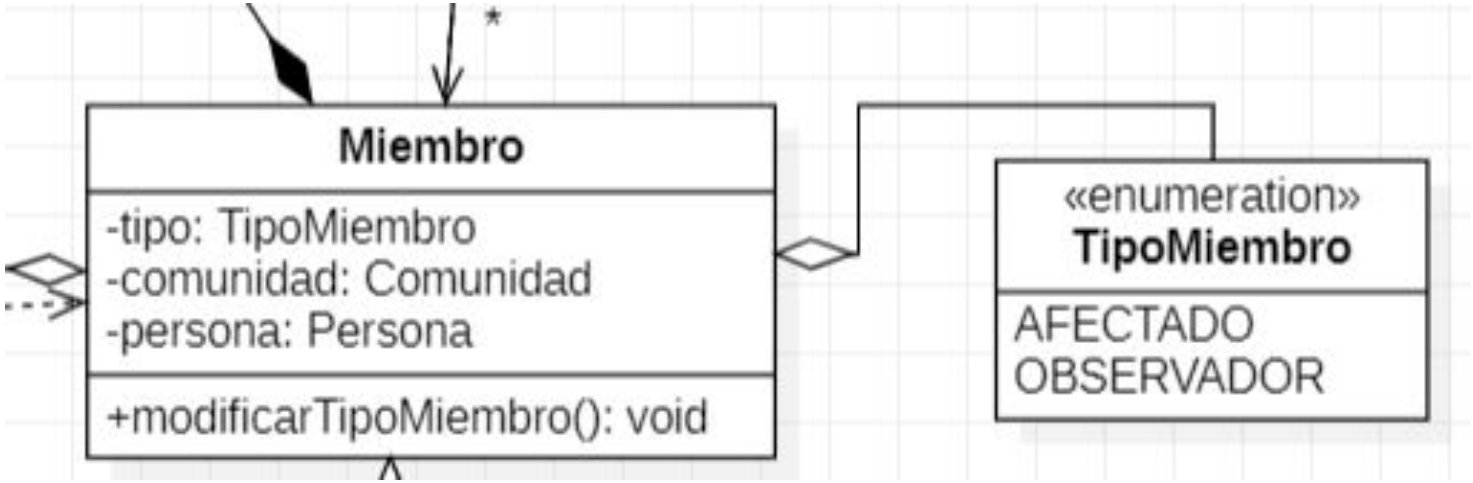
```java
@Override
public void gestionarInicidente(Persona persona, Incidente incidente) {
  Pair<Incidente, Persona> par = Pair.of(incidente, persona);
  listaPares.add(par);
}
private Boolean dentroDeLasUltimas24Horas(Incidente incidente) {
  LocalDateTime fechaCierre = incidente.getFechaCierre();
  if(fechaCierre != null) {
    return fechaCierre.isAfter(LocalDateTime.now().minusHours(24));
  }
  LocalDateTime fechaApertura = incidente.getFechaApertura();
  return fechaApertura.isAfter(LocalDateTime.now().minusHours(24));
}

public void notificarIncidentes(Persona personaANotificar) {
  List<Pair<Incidente, Persona>> paresDeIncidentes = listaPares.stream().filter(par ->
      par.getRight().getEmail().equals(personaANotificar.getEmail())).toList();
  List<Incidente> incidentes = paresDeIncidentes.stream().map(Pair::getLeft).toList();
  incidentes.forEach(incidente -> {
    if (dentroDeLasUltimas24Horas(incidente)) {
      String mensaje = incidente.generarMensaje();
      Notificador.obtenerInstancia().enviarNotificacion(personaANotificar, mensaje); // porque es un Singleton
    }
  });
  listaPares.removeAll(paresDeIncidentes);
}
```

Miembros Afectados/Observadores

Diseño de la condición de Afectado u Observador del Miembro

Afectado/Observador – Diseño general

Un Miembro de una comunidad puede ser Afectado/Observador. por ahora no hay una funcionalidad determinada, implementamos el *tipo* de miembro con un enum. Esto puede cambiar a futuro.



Ranking de Incidentes

Diseño del Módulo Rankeador de Incidentes; diferentes rankings solicitados.

Rankings – Diseño general

Implementamos una entidad **Rankeador** que genera tres tipos de rankings:

1. Entidades con mayor promedio de tiempo de cierre de incidentes
2. Entidades con mayor cantidad de incidentes reportados en la semana
3. Mayor grado de impacto de las problemáticas

Estos métodos reciben por parámetro la lista de entidades a analizar.

Relacionamos a **Entidades** con **Incidentes** de la siguiente forma:

Una *entidad* tiene una lista de *establecimientos*. Un *establecimiento* tiene una lista de *prestaciones*. Una *prestación* tiene una lista de *incidentes*.

Entidad → Establecimientos → Prestaciones → Incidentes

A partir de ello, creamos los **rankings** que ordenan una lista de entidades según los distintos criterios que cumplan sus incidentes asociados, exceptuando el ranking 3, que ordena una lista de incidentes relacionados a una lista de entidades.

Rankings – Diseño general

La clase **Incidente**, tiene además métodos que facilitan la jerarquización de los rankings.

```
public Duration calcularTiempoCierre() {
  //IMPLEMENTACION
}
```

Calcula el tiempo que transcurre entre que se abre y se cierra el incidente

```
public boolean perteneceSemanaActual() {
  //IMPLEMENTACION
}
```

Checkea si el incidente fue creado en la semana actual

```
public int calcularImpactoSobreComunidad()
{
  //IMPLEMENTACION
}
```

Calcula el impacto que genera en la comunidad en base a cuantos miembros hay en ella

```
public boolean esReciente(){
  // IMPLEMENTACION
}
```

Checkea si el incidente fue creado en las últimas 24 hs.

Rankings – Entidades con mayor promedio de tiempo de cierre de incidentes

Este método de elaboración de ranking recibe una lista de *entidades* y devuelve una lista de *cadenas de caracteres* con las denominaciones de las entidades.
Cada entidad es capaz de obtener su promedio de cierre de incidentes, en base a todos los incidentes que tiene asociados. Cabe aclarar que, cada incidente calcula su tiempo de cierre individualmente.

```
// ENTIDAD
public long obtenerPromedioCierreIncidentes() {
  List<Incidente> incidentes = this.obtenerIncidentesSemanales().stream().
      filter(incidente -> !incidente.isAbierto()).toList();
  long totalSegundos = incidentes.stream().mapToLong(incidente -> incidente.calcularTiempoCierre()
      .toSeconds()).sum();
  long cantidadIncidentes = incidentes.size();
  if (cantidadIncidentes == 0)
  {
    return 0;
  }
  return totalSegundos / cantidadIncidentes;
}
```

```
// RANKEADOR
public List<String> elaborarRankingPromedioCierre(List<Entidad> entidades) {
  return entidades.stream()
      .sorted(Comparator.comparingLong(Entidad::obtenerPromedioCierreIncidentes).reversed())
      .toList().stream().map(Entidad::getDenominacion).toList();
}
```

Rankings – Entidades con mayor cantidad de incidentes reportados en la semana

Cada entidad es capaz de obtener sus incidentes semanales.
Este método de elaboración de ranking recibe una lista de *entidades* y devuelve una lista de *cadenas de caracteres* con las denominaciones de las entidades.

```
// ENTIDAD
public List<Incidente> obtenerIncidentesTotales() {
  return establecimientos.stream()
      .flatMap(establecimiento -> establecimiento.obtenerIncidentesTotales()
          .stream()).collect(Collectors.toList());
}
public List<Incidente> obtenerIncidentesSemanales() {
  return this.obtenerIncidentesTotales().stream().
      filter(Incidente::perteneceSemanaActual).toList();
}
```

```
// RANKEADOR
public List<String> elaborarRankingCantidadIncidentesReportados(List<Entidad> entidades){
  return entidades.stream()
      .sorted(Comparator.comparingInt((Entidad entidad) -> entidad.obtenerIncidentesSemanales() List<Incidente>
          .stream().filter(incidente -> (!incidente.esReciente() && !incidente.isAbierto())) Stream<Incidente>
          .toList().size()).reversed()).toList().stream().map(Entidad::getDenominacion).toList();
}
```

Rankings – Mayor grado de impacto de las problemáticas

Este método de elaboración de ranking recibe una lista de *entidades* y devuelve una lista de *cadenas de caracteres* con las denominaciones de los incidentes.

Como actualmente no está desarrollado completamente el criterio de este ranking, por ahora el impacto que tiene un incidente depende de la cantidad de miembros de la comunidad afectada por el incidente. Esto puede cambiar

```
// INCIDENTE
public int calcularImpactoSobreComunidad()
{
  return this.getComunidad().getMiembros().size();
}
```

```
//RANKEADOR
public List<String> elaborarRankingGradoImpactoProblematicas(List<Entidad> entidades){
  return entidades.stream().flatMap(entidad -> entidad.obtenerIncidentesSemanales().stream())
      .toList().stream().sorted(Comparator.comparingInt(Incidente::calcularImpactoSobreComunidad)
          .reversed()).toList().stream().map(Incidente::getDenominacion).toList();
}
```

Informes

Generación (y envío) de informes para Entidades Prestadoras y Organismos de Control

Informes – Diseño general

Implementamos una entidad *singleton* **GeneradorDeInformes** que se encarga de crear los informes semanalmente. También utilizamos un *adapter* por si a futuro se decide cambiar la herramienta con la cual se generan los informes.

Actualmente utilizamos *ApachePDFBox*

```
public class GeneradorDeInformes {
 private Rankeador rankeador;
 private PDFAdapter adapter;
              //
              //
              //
public void generarInforme(EntidadPrestadora entidadPrestadora) {

  List <Entidad> entidades = entidadPrestadora.getEntidades();
  List<String> rankingPromedioCierre = Rankeador.obtenerInstancia().elaborarRankingPromedioCierre(entidades);
  List<String> rankingCantidadIncidentes = Rankeador.obtenerInstancia().elaborarRankingCantidadIncidentesReportados(entidades);
  List<String> rankingMayorImpacto = Rankeador.obtenerInstancia().elaborarRankingGradoImpactoProblematicas(entidades);

  this.adapter.generarInforme(entidadPrestadora.getDenominacion(), rankingPromedioCierre,
      rankingCantidadIncidentes, rankingMayorImpacto);
}
```

Distribución de trabajo en el equipo

Comenzamos tomando decisiones generales de diseño sobre la entrega en su totalidad. A raíz de ello, dividimos los requerimientos entre los cinco y los planteamos con mayor grado de detalle. A continuación, fuimos completando en grupo cada uno de los puntos. De esta manera logramos estar todos involucrados en las distintas secciones de la presente entrega.

Fin y Gracias

¿Preguntas?

Equipo 4

Integrantes:

- Carriquiry Castro, Joaquin
- Lamas, Chabela María
- Montenegro Aguilar, Gabriel Montenegro
- Pérez Gribnicow, Irina Maia
- Sangroni, Luciano Gabriel