Schedule Generator

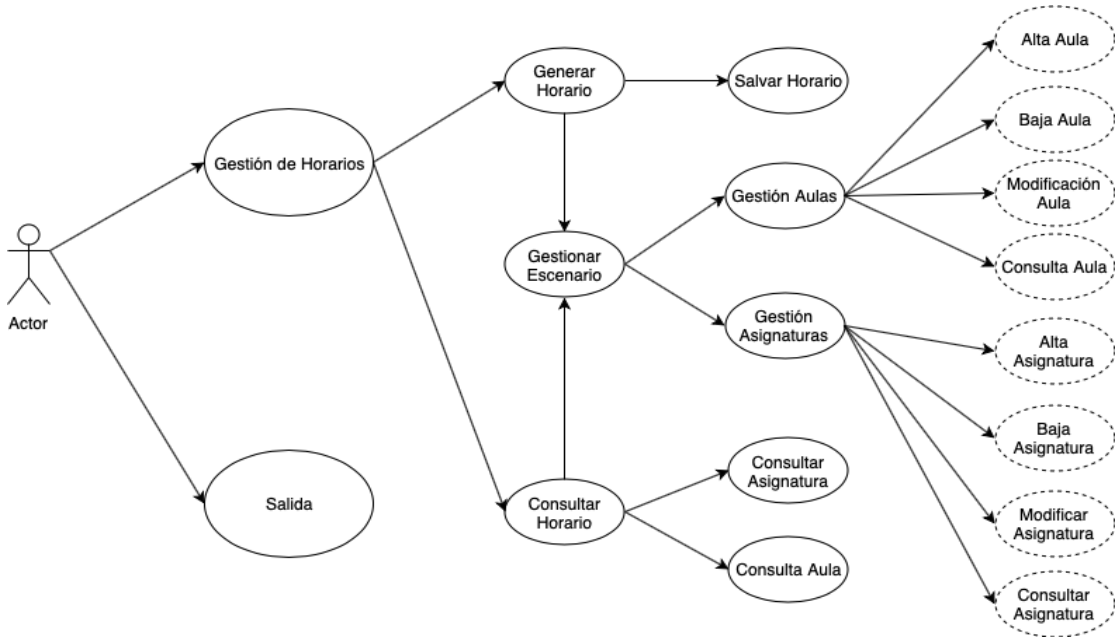
16 de Noviembre de 2018

DEFINICIÓN CASOS DE USO

Nuestros objetivos a cumplir para esta práctica es poder ofrecer un generador de algoritmos cuyo caso de uso principal sea la gestión de horarios, pudiendo permitir además de dicha generación la gestión del escenario asociado, es decir, las aulas y las asignaturas.

De aquí se desprende los siguientes casos y subcasos de uso:

- **Gestión de Horario**: Se ofrece al usuario la posibilidad de escoger entre generar un horario a partir de unos archivos de datos, gestionar un escenario o consultar un horario guardado.
    - **Generar horario**: Genera el horario y ofrece al usuario la posibilidad de guardarlo.
    - **Gestión de escenario**: Se ofrece al usuario la posibilidad de gestionar las aulas (alta, baja, modificación o consulta) y las asignaturas (alta, baja, modificación y consulta).
    - **Consultar Horario**: Cargar un horario previamente creado.
- **Salir**.



MODELO CONCEPTUAL DE DATOS

Adjuntamos el modelo conceptual de datos en un fichero a parte. En adelante se describen los atributos y métodos de las clases representadas en dicho modelo UML:

Classroom

`public abstract class Classroom{}`

Representa el concepto de aula.

Atributos:

`name`: identifica el aula (ejemplos: A6201, 506, AD4).

`capacity`: indica la capacidad máxima del aula.

`type`: indica el tipo de aula del objeto (LABORATORY o THEORY).

`multimedia`: indica si el aula dispone de equipo multimedia o audiovisual.

Métodos:

`public Classroom()`

Constructor vacío

`public Classroom(String n, int cap, UtilsDomain.ClassType t, boolean m)`

Constructor básico
n : nombre del aula
cap : capacidad del aula
t : tipo de aula
m : si el aula tiene multimedia(true) o no(false)

`public Classroom(Vector<String> parse)`

Constructor por Strings (para la comunicación entre capas)
parse : atributos del aula en formato String

`public static Classroom fromStr(Vector<String> c)`

Llama al constructor por String del hijo correspondiente según el atributo type
c : atributos del aula en formato String

`public String getName()`

Getter del atributo name
Retorna un String que es el nombre del aula

`public void setName(String name)`

Setter del atributo name
name : Nombre que asignaremos al aula

`public int getCapacity()`

Getter del atributo capacity
Retorna un int que es la capacidad máxima del aula

`public void setCapacity(int capacity)`

Setter del atributo capacity
capacity : Capacidad máxima que asignaremos al aula

`public UtilsDomain.ClassType getType()`

Getter del atributo type
Retorna un ClassType que indica el tipo del aula

`public void setType(UtilsDomain.ClassType type)`

Setter del atributo type
type : indica el tipo que le asignaremos al aula

`public boolean isMultimedia()`
Getter del atributo multimedia
Retorna un booleano que indica si aula dispone de sistema multimedia

`public void setMultimedia(boolean multimedia)`
Setter del atributo multimedia
multimedia : indica si el aula dispone de sistema multimedia

`public Vector<String> toStr()`
Convierte el objeto en un vector de Strings
Retorna un vector con los atributos del aula en formato String

TheoryClassroom

`public class TheoryClassroom extends Classroom {}`
Representa aulas de Teoría
Atributos:
Métodos:
`public TheoryClassroom()`
Constructor vacío

`public TheoryClassroom(String n, int cap, boolean m)`
Constructor básico
`name:` nombre del aula de teoría
`capacity:` capacidad del aula de teoría
`multimedia:` si el aula de teoría dispone de equipo multimedia

`public TheoryClassroom(Vector<String> parse)`
Constructor por String
`parse:` Vector de String que contiene los atributos del aula

`public Vector<String> toStr()`
Transforma el objeto en un Vector de Strings
Retorna un Vector con los atributos del aula en formato String

LabClassroom

`public class LabClassroom extends Classroom{}`

Representa aulas de Laboratorio

Atributos:

Métodos:

`numComputers` : Número de computadores de que dispone el aula

`public LabClassroom()`

Constructor vacío

`public LabClassroom(String name, int capacity, boolean multimedia, int nComputers )`

`Constructor básico`

`name`: nombre del aula de teoría

`capacity`: capacidad del aula de teoría

`multimedia`: si el aula de teoría dispone de equipo multimedia

`public LabClassroom(Vector<String> parse)`

Constructor por String

`parse`: Vector de String que contiene los atributos del aula

`public int getNumComputers()`

Getter del atributo numComputers

Retorna un int con el número de computadores de los que dispone el aula

`public void setNumComputers(int numComputers)`

Setter del atributo numComputers

`numComputers`: numero de computadores que asignaremos al aula

`public int realCapacity(int ppc)`

Calcula la capacidad real del aula si hay grupos por ordenadores

Retorna el máximo número de personas en el aula para que nadie se quede sin computador

`public Vector<String> toStr()`

Transforma el objeto en un Vector de Strings

Retorna un Vector con los atributos del aula en formato String

ClassroomSet

`public class ClassroomSet{}`

Representa un conjunto de aulas de Teoría y Laboratorio

Atributos:

`theoryClassroomSet`: HashMap que contiene todas las aulas de teoría.

`labClassroomSet`: HashMap que contiene todas las aulas de Laboratorio.

Métodos:

`public ClassroomSet()`

Constructr vacío

`public ClassroomSet(ArrayList<Classroom> cc)`

Constructor básico

`cc`: ArrayList con distintos valores de aulas

`public ClassroomSet(ArrayList<TheoryClassroom> theory, ArrayList<LabClassroom> lab)`

Constructor separando teoría y laboratorio

`theory`: Arraylist con distintos valores de aulas de teoría

`lab`: Arraylist con distintos valores de aulas de laboratorio

`public ClassroomSet(Vector< Vector<String> > vec)`

Cosntructora por String

`vec`: contiene los atributos de la clase en formato String

`private boolean labExists(String name)`

Comprueba si existe un aula de laboratorio concreta

`name`: nombre del aula que queremos comprobar si existe

Retorna un booleano que indica si existe el aula de laboratorio `name` o no

`private boolean theoryExists(String name)`

Comprueba si existe un aula de teoría concreta

`name`: nombre del aula que queremos comprobar si existe

Retorna un booleano que indica si existe el aula de teoría `name` o no

`private LabClassroom getLabClassroom (String name)`

Retorna un aula de laboratorio concreta

`name`: nombre del aula de laboratorio que queremos

Retorna el aula de laboratorio `name`

`private TheoryClassroom getTheoryClassroom (String name)`

Retorna un aula de teoría concreta

`name`: nombre del aula de teoría que queremos

Retorna el aula de teoría `name`

`private ArrayList<Classroom> classroomUnion(ArrayList<Classroom> cc1, ArrayList<Classroom> cc2)`

Une dos ArrayLists

`cc1`: Arraylist de aulas 1

`cc2`: Arraylist de aulas 2

Retorna un Arraylist que es la unión de cc1 y cc2

`public ArrayList<Classroom> getClassroomValues()`

Getter de los valores de aulas

Retorna un Arraylist con todas las aulas

`public void addClassroomSet(ArrayList<Classroom> cc)`

Añade una serie de aulas a nuestro conjunto actual

`cc`: Arraylist de aulas que queremos añadir

`public ArrayList<TheoryClassroom> getTheoryClassroomSet()`

Getter de los valores del atributo theoryClassroomSet

Retorna un Arraylist con las aulas de teoría

`public void addTheoryClassroomSet(ArrayList<TheoryClassroom> theory)`

Añade una serie de aulas de teoría a nuestro conjunto actual

`theory`: Arraylist de aulas de teoría que queremos añadir

`public ArrayList<LabClassroom> getLabClassroomSet()`

Getter de los valores del atributo labClassroomSet

Retorna un Arraylist con las aulas de laboratorio

`public void addLabClassroomSet(ArrayList<LabClassroom> lab)`

Añade una serie de aulas de laboratorio a nuestro conjunto actual

`theory`: Arraylist de aulas de laboratorio que queremos añadir

`public int getNumClassrooms()`

Retorna el número de aulas que tenemos en total

`public boolean exists (String name)`

Indica si existe un aula concreta

`name`: aula que debemos comprobar si existe

Retorna un booleano indicando si el aula `name` existe

`public UtilsDomain.ResultOfQuery getClassroom(String name)`

Obtiene un aula concreta

`name`: aula que debemos obtener

Retorna un ResultOfQuery. Si el primer valor es true, el segon valor contendrá el aula `name`. Si es false el segundo valor es irrelevante.

`public Vector< Vector< String> > toStr()`

Transforma el objeto en un Vector de Strings

Retorna un Vector con los atributos del conjunto de aulas en formato String

ClassroomSession

`public class ClassroomSession{}`

Representa un conjunto de pares de aulas y sesiones

Atributos:

`classroomSessionSet: ArrayList de Pares de aulas y sesiones`

Métodos:

`public ClassroomSession()`

Constructor vacío

`public ClassroomSession(ClassroomSet crSet)`

Constructor básico

`crSet`: conjunto de aulas a partir de las cuales generaremos los pares aula-sesión

`public ClassroomSession(ClassroomSession cs)`

Constructor por copia

`cs`: Objeto ClassroomSession que debemos replicar

`public ArrayList<UtilsDomain.Pair> getClassroomSessionSet()`

Getter del atributo classroomSessionSet

Retorna el atributo classroomSessionSet

`public void setClassroomSessionSet(ArrayList<UtilsDomain.Pair> classroomSessionSet)`

Setter del atributo classroomSessionSet

`classroomSessionSeT`: valor que asignaremos al atributo

`public UtilsDomain.Pair getPair(int index)`

Obtiene un pair aula-sesión concreto

`index`: índice de classroomSessionSet donde se encuentra el pair que debemos devolver

Retorna el Pair que se encontraba en la posicion `index`

`public int size()`

Retorna el número de pares aula-sesión que tenemos

`public boolean delete(int index)`

Elimina un pair aula-sesión concreto

`index`: índice de classroomSessionSet donde se encuentra el pair que debemos eliminar

Retorna un booleano indicando si se ha ejecutado correctamente la eliminación

Schedule

`public class Schedule{}`

Representa un horario

Atributos:

`classroomFile`: nombre del fichero desde el que se han importado las aulas del horario

`subjectFile`: nombre del fichero desde el que se han importado las assignaturas del horario

`correct`: valor que utlizamos en el backtracking para descartar horarios

`timetable`: Contiene todas las unidades mínimas del horario.

Métodos:

`public Schedule()`

Constructor vacío

`public Schedule(String classroomFile, String subjectFile)`

Constructor simple

`classroomFile`: nombre del fichero desde el que se han importado las aulas del horario

`subjectFile`: nombre del fichero desde el que se han importado las assignaturas del horario

`public Schedule(String classroomFile, String subjectFile, HashMap<String, ArrayList<MUS>> timetable)`

Constructor básico

`classroomFile`: nombre del fichero desde el que se han importado las aulas del horario

`subjectFile`: nombre del fichero desde el que se han importado las assignaturas del horario

`timetable`: Contiene todas las unidades mínimas del horario.

`public Schedule(Schedule sched)`

Constructora por copia

`sched`: objeto horario que debemos replicar

`private void addOrdered(ArrayList<MUS> v, MUS mus)`

Se encarga de añadir `mus` de forma ordenada dentro de `v`

`v`: Arraylis con MUSes de una misma assignatura

`mus`: MUS de la misma assignatura que debemos añadir

`private int findPosition(ArrayList<MUS> v, MUS mus)`

Busca el índice dentro de `v` donde se encuentra `mus`

`v`: Arraylis con MUSes de una misma assignatura

`mus`: MUS de la misma assignatura del que debemos encontrar el índice

Retorna el índice de `mus` dentro de `v`

`public String getClassroomFile()`

Getter del atributo classroomFile
Retorna el atributo classroomFile

`public void setClassroomFile(String classroomFile)`

Setter del atributo classroomFile
`classroomFile:` valor que debemos asignar al atributo classroomFile

`public String getSubjectFile()`

Getter del atributo subjectFile
Retorna el atributo subjectFile

`public void setSubjectFile(String subjectFile)`

Setter del atributo subjectFile
`subjectFile:` valor que debemos asignar al atributo subjectFile

`public boolean isFail()`

Getter del atributo correct
Retorna el atributo correct negado(por motivos de usabilidad)

`public void setFail(boolean correct)`

Setter del atributo correct
`correct:` valor que debemos asignar al atributo correct

`public void fail()`

Setter a false por defecto del atributo correct (por motivos de usabilidad)

`public HashMap<String, ArrayList<MUS>> getTimetable()`

Getter del atributo timetable
Retorna el atributo timetable

`public void setTimetable(HashMap<String, ArrayList<MUS>> timetable)`

Setter del atributo timetable
`timetable:` valor que debemos asignar al atributo timetable

`public boolean isEmpty()`

Retorna un booleano indicando si el atributo timetable está vacío

`public int size()`

Retorna un int con el número total de unidades mínimas horarias que contiene el atributo timetable

`public void add(MUS mus)`

Añade un MUS al atributo timetable

`mus:` objeto que debemos añadir

`public boolean delete(MUS mus)`

Elimina un MUS concreto del timetable

`mus:` objeto que queremos eliminar

`R`etorna un booleano indicando si se ha podido eliminar

`public ArrayList<MUS> unset()`

Transforma el atributo timetable en un ArrayList de MUSes

Retorna el ArrayList de MUSes

`public boolean valid()`

Nos indica si el timetable actual cumple las restricciones asignadas.

Subject

`public class Subject{}`

Representa las diferentes asignaturas de un horario.

Atributos:

`name`: nombr de la asignatura.

`numberStudents`: número máximo de estudiantes de la asignatura.

`level`: nivel al que corresponde.

`hoursClasses`: vector con horas de teoría, de laboratorio y de problemas.

`numberOfGroups`: vector con el número de grupos y el número de subgrupos.

`tyShift`: tipo de turno de la asignatura (Mañana, Tarde o Ambos).

Métodos:

`public Subject()`

Constructor vacío

`public Subject(String name, int numberStudens, int level, int[] hoursClasses, int[] numberOfGroups)`

Constructor básico
name : nombre de la asignatura
number students
level: nivel de la asignatura
hoursClasses: distribución de las horas de clase entre los tipos.
numberOfGrous: numero de grupos y subgrupos.

`public Subject(Vector<String> vectorMembers)`

Constructor por Strings (para la comunicación entre capas)
parse : atributos del Subject en formato String

`public void setName(String name)`

Setter del atributo name.
name: Nombre que se asigna a la asignatura.

`public String getName()`

Getter del atributo name
Retorna un String que es el nombre del aula

`public void setNumberStudents(int numberStudents)`

Setter del atributo name
numberStudents: número de estudiantes que se asignará a la asignatura.

`public int getNumberStudents()`

Getter del número de estudiantes.
Devuelve el número de estudiantes.

`public void setLevel(int level)`

Setter del atributo level
level: Nivel que se asigna a la asignatura.

`public int getLevel()`

Getter del atributo level.
Devuelve el atributo level.

`public void setHoursClasses(int theoryHours, int laboratoryHours, int problemsHours)`

Setter de las diferentes horas de clase.
theoryHours: Horas de teoría.
laboratoryHours: Horas de laboratorio.
problemsHours: Horas de problemas.

`public int[] getHoursClasses()`

Getter de las diferentes horas de clase.
Devuelve un array con los diferentes tipos.

`public int getTheoryHours()`

Getter de las horas de teoría.
Devuelve las horas de teoria.

`public int getLaboratoryHours()`

Getter de las horas de laboratorio.
Devuelve las horas de laboratorio.

`public int getProblemsHours()`

Getter de las horas de problemas.
Devuelve las horas de problemas..

`public void setNumberOfGroups(int groups, int subgroups)`

Setter del número de grupos y subgrupos.

`public int[] getNumberOfGroups()`

Getter del número de grupos y subgrupos.
Devuelve array con el número de grupos y subgrupos.

`public void setTypeShift(typeShift tyShift)`

Setter del tipo de turno..

`public Vector<String> toStr()`

Devuelve el objeto en forma de vector de strings (uno por cada parámetro).

SubjectsSet

`public class SubjectsSet{}`

Representa un conjunto de asignaturas.

Atributos:

`set`: HashMap que representa el set de asignaturas, siendo la key el nombre de la asignatura.

Métodos:

`public SubjectsSet()`

Constructor vacío

`public SubjectsSet(ArrayList<Subject> subjects)`

Constructor a partir de Arraylist de Subjects.
subjects: Array List con el nombre de las asignaturas como key.

`public SubjectsSet(Vector< Vector<String> > subjectsSet)`

Constructor del conjunto de asignaturas en formato string.
subjectsSet: asignaturas en formato string.

`public setSet(ArrayList<Subject> subjects)`

Pone en el set todas las asignaturas del array.
subjects: Array de asignaturas para ser añadidas.

`public ArrayList<Subject> unset()`

Devuelve el set en formato ArrayList de Subjects.

`public Vector< Vector<String> > toStr()`

Convierte el set en un conjunto de asignaturas representadas como strings.

`public Subject getSubject(String name)`

Obtiene una asignatura a partir de su nombre.
name: nombre de la asignatura a recuperar.

`public boolean putSubject(Subject s)`

Introduce una nueva asignatura en el set.
s: Asignatura a introducir.

`public boolean popSubject(String name)`

Elimina una asignatura del set.

name: nombre de la asignatura a eliminar.

`public int length()`

Devuelve el tamaño del set.

`public boolean belongs(String s)`

Devuelve true si la asignatura con nombre s existe en el set.
s: nombre de la asignatura a buscar.

`public static boolean compare(Subjects s1, String op, Subjects s2`

Devuelve true si el operador booleano entre los dos subjects se evalúa a true.
s1: asignatura 1
op: operador a aplicar.
s2: asignatura 2

`public static subjectsSort(ArrayList<Subject> set)`

Ordena las asignaturas empleando Mergesort.
set: asignaturas a ordenar.

CtrlScheduleGeneration

`public class CtrlScheduleGeneration{}`

Genera el horario a partir del escenario proporcionado por el CtrlDomain.
Atributos:

`schedule`: Horario definitivo una vez generado.
`classroomSession`: Dominio general para las variables.
`vars`: Conjunto de variables a las que asignar valores dentro del dominio.

Métodos:

`public CtrlScheduleGenerator(String crFile, String sFile)`

Constructor que únicamente recibe el nombre de los escenarios.

`public Schedule generateSchedule(LinkedList<MUS> vars, ClassroomSession classroomSession)`

Inicializa el escenario para la generación del horario y lanza la ejecución de este.
vars: conjunto de variables a las que asignar valor.
classroomSession: dominio para las variables.
Devuelve el horario una vez generado.

`private void filterUnaryConstraints(LinkedList<MUS> vars)`

Filtra el dominio de las variables para que cumplan las restricciones unarias.
vars: variables a las que se debe filtrar el dominio.

`public Schedule chronologicalBacktracking(LinkedList<MUS> futureVars, Schedule solution)`

Algoritmo de satisfacción de restricciones que va generando el horario mediante la asignación de valores a las variables.
futureVars: variables que no tienen valor del dominio asignado aun.
solution: solución parcial de la asignación de valores.
Devuelve el horario una vez generado.

MUS

`public class MUS{}`

Unidad mínima de horario. Almacena la sesión, el aula y el grupo correspondiente.
Atributos:

`classclass`: Grupo del MUS:
`classroom`: Aula de MUS.
`session`: Sesión del MUS.
`domain`: Dominio de la variable.

Métodos:

`public MUS()`

Constructor vacío.

`public MUS(ClassClass classclass, Classroom classroom, Session session)`

Constructor con los atributos de la clase.

`public MUS(ClassClass classclass, UtilsDomain.Pair<Classroom, Session>`

Constructor con los atributos de la clase pero con Classroom y Session como Pair.

`public MUS(Vector< Vector<String> > mus)`

Constructor a partir de Strings.

`public ClassClass getClassClass()`
Devuelve la Classclass del MUS.

`public Classroom getClassroom()`
Devuelve la classroom del MUS.

`public Session getSession()`
Devuelve la session del MUS.

`public void setClassClass(ClassClass classclass)`
Setter de classclass.

`public void setClassroom(Classroom classroom)`
Setter de classroom

`public void setSession(Session session)`
Setter de session.

`public UtilsDomain.Pair<Classroom, Session> getClassroomSessionPair()`
Getter de Classroom y Session como Pair.

`public Subject getSubject()`
Getter de Subject, que se encuentra en la Classclass.

`public ClassroomSession getDomain()`
Getter del domain del MUS.

`public void setDomain(ClassroomSession domain)`
Setter del Domain.

`public int domainSize()`
Devuelve el tamaño del domain.

`public void assign(UtilsDomain.Pair<Classroom, Session> csPair)`
Asigna a la variable el par classroom-session.
csPair: Par classroom-session.

```
public UtilsDomain.Pair<Classroom, Session>
getValueDomain(int i)
```
Getter de un valor dado del dominio.
i: identificador del elemento dentro del dominio.

```
public void deleteFromDomain(int i)
```
Elimina un elemento del dominio.
i: identificador del elemento dentro del dominio.

```
public Vector< Vector<String> > toStr()
```
Convierte el MUS en un vector de strings.

ClassClass

```
public abstract class ClassClass
```
Representa el concepto de Clase o conjunto de estudiantes que comparten un mismo horario.

Atributos:

`group`: Número de grupo (10, 20, 30)
`identifier`: Identificador único de la clase (ASO10, ASO11, PAR 20)
`subject`: Asignatura de la clase
`type`: Tipo de clase, THEORY, LABORATORY o PROBLEMS
`quantityStudents`: Cantidad de estudiantes en esta clase (10, 13,55)
`shift`: Turno en la que se imparte la clase

Métodos:

```
public ClassClass(String identifier, Subject subject, int group,
int quantityStudents, UtilsDomain.typeShift shift,
UtilsDomain.ClassType type)
```
Constructor con los atributos de la clase

```
public ClassClass( Vector<String> myStringVector,
UtilsDomain.ClassType type)
```
Constructor a partir de String, y obteniendo el tipo de clase como atributo

```
public void setGroup(int group)
```

Setter del atributo group.

```
public void setIdentifier(String identifier)
```
Setter del atributo identifier

```
public int getGroup()
```
Getter del atributo group

```
public String getIdentifier()
```
Getter del atributo Identifier

```
public void setSubject(Subject subject)
```
Setter del atributo Subject

```
public Subject getSubject()
```
Getter del atributo Subject

```
public UtilsDomain.ClassType getType()
```
Getter del atributo type

```
public UtilsDomain.typeShift getShift()
```
Getter del atributo shift

```
public void setShift(UtilsDomain.typeShift shift)
```
Setter del atributo shift

```
public int getQuantityStudents()
```
Getter del atributo Students

```
public void setQuantityStudents(int quantityStudents)
```
Setter del atributo quantityStudents

```
public abstract void setSubGroup(int subGroup);
```
Setter del atributo subGroup, este método es abstracto por lo que solo la pueden implementar los hijos.

```
public abstract int getSubGroup();
```
Getter del atributo SubGroup, esté método es abstracto por lo que solo lo pueden implementar los hijos de la clase.

```
public abstract Vector<String> toStr();
```
Devuelve el objeto en forma de vector de strings (uno por cada parámetro).

```
public static ClassClass fromStr( Vector<String> c)
```

Devuelve el objeto creado a partir de un vector de string de donde este vector tiene un atributo por posición.

TheoryClass

```
public class TheoryClass extends ClassClass
```

Clase hija de la clase abstract ClassClass.

Atributos:
`subGroup`: Número del subgrupo, como es teoría siempre será un múltiplo de 10 (10, 20, 30)

Métodos:

```
public TheoryClass(String identifier, Subject subject, int
group, int quantityStudents, UtilsDomain.typeShift shift, int
subGroup)
```
Constructora de la clase, conc os atributos de la clase. Este método recibe los atributos, los cuales pasa al super, cambiando el type por el tipo de la clase actual (THEORY)

```
public TheoryClass( Vector<String> myStringVector )
```
Constructor a partir de un vector de sting, donde por cada posición tenemos un atributo.

```
public void setSubGroup(int subGroup)
```
Setter del atributo SubGroup, este método a sido @Override del método padre, ya que el hijo es el que tiene este atributo por lo que el padre no tiene acceso a él.

```
public int getSubGroup()
```
Getter del atributo SubGroup, este método a sido @Override del método padre, ya que el hijo es el que tiene este atributo por lo que el padre no tiene acceso a él.

```
public Vector<String> toStr()
```
Devuelve el objeto en forma de vector de strings (un atributo por cada posición).

LaboratoryClass

`public class LaboratoryClass extends ClassClass`

Clase hija de la clase abstract ClassClass.

Atributos:
`subGroup`: Número del subgrupo (11, 22, 33)

Métodos:

`public LaboratoryClass(String identifier, Subject subject, int group, int quantityStudents, UtilsDomain.typeShift shift, int subGroup`
Constructora de la clase, conc os atributos de la clase. Este método recibe los atributos, los cuales pasa al super, cambiando el type por el tipo de la clase actual (LABORATORY)

`public LaboratoryClass( Vector<String> myStringVector )`
Constructor a partir de un vector de sting, donde por cada posición tenemos un atributo.

`public void setSubGroup(int subGroup)`
Setter del atributo SubGroup, este método a sido @Override del método padre, ya que el hijo es el que tiene este atributo por lo que el padre no tiene acceso a él.

`public int getSubGroup()`
Getter del atributo SubGroup, este método a sido @Override del método padre, ya que el hijo es el que tiene este atributo por lo que el padre no tiene acceso a él.

`public Vector<String> toStr()`
Devuelve el objeto en forma de vector de strings (un atributo por cada posición).

ProblemsClass

`public class ProblemsClass extends ClassClass`
Clase hija de la clase abstract ClassClass.

Atributos:

`subGroup`: Número del subgrupo, como es teoría siempre será un múltiplo de 10 (10, 20, 30)

Métodos:

`public ProblemsClass(String identifier, Subject subject, int group, int quantityStudents, UtilsDomain.typeShift shift, int subGroup`

Constructora de la clase, conc os atributos de la clase. Este método recibe los atributos, los cuales pasa al super, cambiando el type por el tipo de la clase actual (PROBLEMS)

`public ProblemsClass( Vector<String> myStringVector )`

Constructor a partir de un vector de sting, donde por cada posición tenemos un atributo.

`public void setSubGroup(int subGroup)`

Setter del atributo SubGroup, este método a sido @Override del método padre, ya que el hijo es el que tiene este atributo por lo que el padre no tiene acceso a él.

`public int getSubGroup()`

Getter del atributo SubGroup, este método a sido @Override del método padre, ya que el hijo es el que tiene este atributo por lo que el padre no tiene acceso a él.

`public Vector<String> toStr()`

Devuelve el objeto en forma de vector de strings (un atributo por cada posición).

ClassSet

`public class ClassSet`

Representa el concepto de aula.

Atributos:
`classSet`: Set de clases, donde la key es nombre de la asignatura más número de grupo, el value es el objeto clase. La key tiene el formato (ASO10, ASO11, PAR30)

Métodos:

`public ClassSet()`

Constructora que crea un set de clases vacío.

`public ClassSet( SubjectsSet subjectsSet )`

Constructora que se encarga de crear un set vacío y luego llamar a la función privada createSetOfClasses la cual se encarga de crear todas las clases para el set de asignaturas dado.

`public ClassSet( Vector< Vector<String> > classS )`

Constructor que crea un set de clases a partir de una matriz de String, donde cada fila es una clase y cada columna un atributo de la clase

`private void createSetOfClasses( SubjectsSet subjects )`

Crea un set de clases a partir de un set de asignaturas, itera cada asignatura y lee sus atributos, con los que decide cuántos grupos va a crear(atributo de la asignatura), luego con este atributo puede calcular cuántos alumnos estarán en cada grupo con la siguiente división (Cantidad de alumnos/grupos) este resultado lo almacena en el atributo de la clase “QuantityStudents”, de igual forma para los subgrupos. Si el turno es de MORNING se crean todos los grupos y subgrupos de mañana, sí es AFTERNOON de igual forma, si es BOTH se genera un grupo por la mañana y otro por la tarde de forma intercalada.

`public boolean existsClass( String subjectName, int subGroup )`

Comprueba si una clase dada existe en el set, recibe como entrada el nombre de la asignatura y el número de subgrupo

`public ClassClass getClass( String name, int subGroup )`

Getter del set para obtener una clase enviando el nombre de la asignatura y número de subgrupo

`public void addClass( String identifier, ClassClass newClass )`

Añade una nueva clase al set de clase, basado en el identifier y valor la nueva clase.

`public int size()`

Retorna el tamaño del set.

`public ArrayList<ClassClass> unset()`

Pasa el HashMap a un array list de clases donde en cada posición tiene una clase.

`public static boolean compare(ClassClass s1, String op, ClassClass s2)`

Compara dos objetos de ClassClass pasandole un operador (<, >, <=, >=, !=, ==)

`public Vector< Vector<String> > toStr()`

Transforma el set de clases a una matriz de string, donde cada file es una clase y cada columna un atributo de la clase.

Session

`public class Session`

Representa una sesión, que es un día de la semana y una hora especifica.

Atributos:

`hoursPerDay`: Variable final y static que permite saber la duración de un día en la institución.

`startHour`: Variable final y static que permite saber a qué hora empieza la jornada en la institución.

`daysOfTheWeek`: Variable final y static que permite saber cuántos días de la semana la institución imparte clases, empezando por lunes, ejemplo si el valor fuera 1 entonces solo se imparten clases los lunes.

`hour`: Hora del día entre startHour y startHour+hoursPerDay

`day`: Día de la semana (MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY)

Métodos:

`public Session()`

Constructora que crea una sesión vacía

`public Session(UtilsDomain.Day day, int hour)`

Constructora que crea una sesión con los atributos day y hora

`public Session( Vector<String> myVector )`

Constructora que crea una sesión a partir de un vector de string, donde cada posición es un atributo.

```
public int getHour()
```

Getter del atributo hora

```
public void setHour(int hour)
```

Setter del atributo hora

```
public UtilsDomain.Day getDay()
```

Getter del atributo day

```
public void setDay(UtilsDomain.Day day)
```

Setter del atributo day

```
public Vector<String> toStr()
```

Pasa la sesión a un vector de string donde cada posición es un atributo

```
public static boolean compare( Session s1, String op, Session s2
)
```

Compara dos objetos sesión, donde se pasa como parámetro un operador, si se cumple retorna true si no false. Valores posibles para Op (<,>,<=,>=,!=,==).

DataManager

```
public class DataManager
```

Capa de datos la cual se encarga de salvar y cargar los datos de ficheros.

Métodos:

```
public DataManager( )
```

Constructora vacia

```
public Vector <Vector< String>> importClassrooms(String
fileName) throws IOException
```

Importa un fichero JSON de Classrooms transformándolo a una matriz, donde cada fila es un classroom y cada columna un atributo de la clase classroom

```
public Vector <Vector< String>> importSubjects(String fileName)
throws IOException
```

Importa un fichero JSON de Subjects transformándolo a una matriz, donde cada fila es un Subjects y cada columna un atributo de la clase Subjects

`public Schedule loadSchedule( int fileNum ) throws IOException`
Carga un horario a partir de un fichero y retorna el horario instanciado.

`public ArrayList<String> listScheduleFiles()`
Lista todos los archivos disponibles donde se ha guardado un horario anteriormente

`public void saveSchedule( String fileName, Schedule schedule ) throws Exception`
Guarda un horario en un fichero a partir del objeto Schedule y lo nombra con el nombre fileName

CtrlDomain

`public class CtrlDomain`
Controlador encargado de la capa de dominio, se encarga de realizar la comunicación entre la capa de datos la capa de presentación y la capa de dominio.

Atributos:

`dManager`: Data manager object
`schedule`: Schedule object
`classroomsSet`: ClassroomSet object
`subjectsSet`: SubjectsSet object
`classSet`: ClassSet object
`classroomSession`: ClassroomSession object
`classroomFile`: Nombre del fichero donde se importaron los classrooms
`subjectFile`: Nombre del fichero donde se importaron los subjects

Métodos:

`public CtrlDomain()`
Constructora que crea un nuevo objeto dataManager

`private boolean importClassroom(String file) throws Exception`
Importa un set de classrooms a partir del fichero con el nombre file y lo guarda en el atributo classroomsSet

`private boolean importSubject(String file) throws Exception`

Importa un set de subjects a partir del fichero con el nombre file y lo guarda en el atributo subjectsSet

```
public boolean createScenario(String classroomFile, String
subjectFile) throws Exception
```

Crea un nuevo escenario para poder generar un horario, inicializando las variables de classSet, ClassroomSession, classroomFile, SubjectFile.

```
public Vector<Vector <String>> showSubject()
```

Se encarga de retornar una matriz de asignatura en formato string

```
public Vector <Vector< String>> showClassroom()
```

Se encarga de retornar una matriz de aulas en formato string

```
public UtilsDomain.ResultOfQuery<Schedule> showSchedule()
```

Se encarga de retornar un horario para ser mostrado

```
public void generateSchedule()
```

Genera un nuevo horario a partir de las variables inicializadas en crear escenario, para esto llama a ctrlScheduleGeneration.

```
public void saveSchedule( String newFileName, Schedule schedule
) throws Exception
```

Hace la conexión con el dataManager para guardar el horario generado en un fichero.

```
public ArrayList<String> listScheduleFiles()
```

Retorna una lista del nombre de los ficheros disponibles para cargar un horario.

```
public void loadSchedule(int fileNum) throws IOException
```

Carga un horario desde un fichero guardado previamente, este horario lo almacena en el atributo schedule.

ESTRUCTURA DE DATOS Y ALGORITMOS EN LAS FUNCIONALIDADES PRINCIPALES

Para el núcleo de la aplicación, es decir, la generación de horarios, hemos realizado la implementación del backtracking cronológico. Este recibe una linkedlist que actúa como queue, la cual contiene las variables a las cuales debe asignar valor.

Dichas variables contienen un atributo fijo, una vez las ha recibido el algoritmo, que define el grupo/subgrupo de dicha unidad mínima de horario (las variables) y un atributo dominio. Este último, es el conjunto de posibles valores, después de haberse filtrado las restricciones unarias, a asignar a la variable (pares aula-sesión).

Las restricciones binarias y n-arias se comprueban dos a dos mediante una función validadora de horario que para cada unidad mínima comprueba que no infrinja ninguna restricción con el resto de unidades del horario.

RELACIÓN DE CLASES Y MIEMBROS DEL GRUPO

Cada miembro del grupo ha realizado las clases asignadas que se enumeran a continuación:

Sergio:
- DataManager
- Session
- ClassSet
- Clase Abstracta ClassClass (TheoryClass, LaboratoryClass, ProblemsClass)
- CtrlDomain

Mireia:
- ClassroomSession
- Clase Abstracta Classroom (TheoryClassroom, LabClassroom)
- ClassroomSet
- Schedule
- CtrlDomain
- CtrlScheduleGeneration
- Constraints

Joaquim:
- Subject
- SubjectsSet
- MUS
- CtrlPresenter
- Constraints
- CtrlScheduleGeneration

RELACIÓN DE LIBRERÍAS EXTERNAS UTILIZADAS

Para esta primera entrega únicamente hemos utilizado la siguiente librería:

- **json-Simple**: Librería java simple que permite realizar codificaciones y decodificación en texto JSON (RFC4627).
  Referencia: https://code.google.com/archive/p/json-simple/

MANUAL DE USO DE LA APLICACIÓN

Proporcionamos un Makefile que permite:

- make “nombreDriver” - Ejecutar de manera interactiva cada uno de los drivers proporcionados (uno para cada clase del dominio).
- make junit - Ejecutar una prueba unitaria para la clase Schedule.
- make maintest - Ejecutar un programa de testeo que permite escoger entre los diferentes drivers y tanto la lectura como la escritura se realizan mediante ficheros ubicados en data/drivers/
- make run - Ejecuta la aplicación, permitiendo realizar los diferentes casos de uso.

Todos estos ejecutables están guiados mediante menús y la solicitud al usuario de que desea hacer.