extensibleclustering::application

<<Interface>>
extensibleclustering:: application::StandardViewInterface
Interface that should be implemented by the StandardView

Responsibilities
+ getSelectedFile(): File
+ setResult(result: String): void
+ getSelectedParserName(): String
+ addParseFileObserver(parseObserver: Observer)
+ addUpdateParsersObserver(updateObserver: Observer)
+ addClusterObserver(clusterObserver: Observer)
+ setParserNameSet(names: HashSet<String>): void
+ setClusteringNameSet(names: HashSet<String>): void
+ setVisualisationNameSet(names: HashSet<String>): void
+ getSelectedVisualisationName(): String

<<Singleton>>
<<View>>
extensibleclustering:: application:: StandardView
Displays the User Interface to the user allowing them to pick a parser to use. This could also be extended to let them choose a clustering algorithm / visualisation method.

Attributes:
- fileSelected: File
- parserNameSelected: String
- clusteringNameSet: HashSet<String>
- parserNameSet: HashSet<String>
- clusterNameSelected: String

Responsibilities
<<constructor>> - ParserView()
+ getSelectedFile(): File
+ setResult(result: String): void
+ getSelectedParserName(): String
+ getInstance(): ParserView
+ addParseFileObserver(parseObserver: Observer): void
+ addUpdateParsersObserver(updateObserver: Observer): void
+ addClusterObserver (clusterObserver: Observer): void
+ setParserNameSet(names: HashSet<String>): void
+ setClusteringNameSet(names: HashSet<String>): void
+ setVisualisationNameSet(names: HashSet<String>): void
+ getSelectedVisualisationName(): String

<<Singleton>>
<<Controller>>
extensibleclustering:: application:: ExtensibleClusteringController
Controller for the application. This will be notified when actions occur in the User Interface and will process them accordingly.

Attributes:
- ecModel: ExtensibleClusteringModel

Responsibilities
<<constructor>> - ExtensibleClusteringController()
+ getInstance(): ExtensibleClusteringController

+ parseFile(parserName: String, fileToParse: File)
Notified a file needs to be parsed. Listens for events on the parse file button

+ loadParserPlugins():
Notified when the parser plugins need to be loaded / reloaded. Listens for either on load or reload button pressed.

+ setStandardView(view: StandardViewInterface): void

+ clusterData(): void
Clusters the data and outputs the result in the view

+ visualiseData(): void
Performs the data visualisation using the selected visualisation method, and opens it in a new window

<<Observer>>

<<Model>>
extensibleclustering:: application:: ExtensibleClusteringModel
Model for the ExtensibleClustering Application. This will retrieve plugins and the results from running them. It will also import the plugins when constructed.

Attributes:
- importedPlugins: ImportedPlugins
- dirHelper: DirectoryHelper
- pluginLoader: PluginLoader

Responsibilities
<<constructor>> + ExtensibleClusteringModel()
+ importPlugins(): void
+ getImportedPlugins(): ImportedPlugins
+ parseFile(parserName: String, fileToParse: File): Position[]
+ clusterData(clusteringName: String, data: Position[]): Path
+ visualiseData(visualisationName: String, file: Path): Scene

Use

---

extensibleclustering::dependencies

<<Interface>>
extensibleclustering:: dependencies::Parser
Interface that a parser will need to implement.

Responsibilities:
+ parseFile(file: File): Position[]
+ getID(): String
+ getDescription(): String
+ getSupportedFileExtensions(): ArrayList<String>

<<Interface>>
extensibleclustering:: dependencies::ClusteringAlgorithm
An interface that needs to be implemented by each clustering algorithm. The clusterData method should return the output file.

Methods:
+ clusterData(Position[] data): Centroid[]
+ getLastOutputFile(): File
+ getName(): String
+ getDescription(): String

<<Interface>>
extensibleclustering:: dependencies::Visualisation
Interface that a visualisation method will need to implement.

Responsibilities:
+ visualiseData (Path: dataFile): Scene
+ getName(): String
+ getDescription(): String

---

extensibleclustering:: dependencies::Position
Stores data for a position. This includes the ID, and the components that make up the vector for the position.

Attributes
- ID: String
- Components: Double[]

Responsibilities
<<constructor>> + Position(ID: String, components: Double[])
+ getID(): String
+ getComponents(): Double[]
+ getDistance(otherPosition: Position): Double,
+ toString(): String
+ getDescription(): String
+ hashCode(): int
+ equals(obj: Object): boolean

Throws

<<exception>>
extensibleclustering:: dependencies:: IncomparableComponentsException
Exception that is thrown when the components of two positions are of different lengths and cannot be compared.

Methods:
<<constructor>> + IncomparabelComponentsException()

---

extensibleclustering:: dependencies::DirectoryHelper
Helps check the required plugin, output etc directories exist and can be used. Can also get the path for them so the person creating the plugin does not have to do it.

+ <<constructor>> + DirectoryHelper()
+ createRequiredDirectories(): void
+ getPluginFolder(): Path
+ getOutputFolder(): Path
+ correctFolderStructureExists(): boolean

---

extensibleclustering:: dependencies::Centroid
Class for representing a single cluster

Attributes
- ID: String
- location: Position
- assignedPositions: Vector<Position>

Responsibilities
<<constructor>> + Clusteroid(Id: String, location: Position)
+ getLocation(): Position
+ getID(): String
+ clearAssignedPositions(): void
+ assignPosition(position: Position): void
+ getAssignedPositions(): Vector<Position>
+ centreCluster(): void
+ toString(): String
+ hashCode(): int
+ equals(obj: Object): boolean

---

extensibleclustering:: dependencies::Metrics
Contains metric functions that can be used in the clustering process. These methods require centroids that have had data assigned to them

+ calinskiHarabasz (clusters: Centroid[], numPositions: Integer): Double
+ withinClusterVariance (clusters: Centroid[]): Double
+ betweenClusterVariance (clusters: Centroid[]): Double

---

extensibleclustering::plugins

extensibleclustering:: plugins::ImportedPlugins
Stores the imported plugins in their different types (Parsers, Clustering Algorithms and Visualisation)

Attributes:
- TreeMap<String, Class> importedParsers
- TreeMap<String, Class> importedClusteringAlgorithms
- TreeMap<String, Class> importedVisualisationMethods
- HashMap<String, List<String>> parserSupportedFileTypes
- HashMap<String, String> importedPluginDescriptions

Responsibilities
<<constructor>> + ImportedPlugins()
+ getImportedParsers(): TreeMap<String, Class<?>>
+ getImportedClusteringAlgorithms(): TreeMap<String, Class<?>>
+ getImportedVisualisationMethods(): TreeMap<String, Class<?>>
+ getNumOfTotalPlugins (): int
+ getNumOfParserPlugins(): int
+ getNumOfClusteringPlugins(): int
+ getNumOfVisualisationPlugins(): int
+ getParserSupportedFileTypes(): HashMap<String, List<String>>
+ getImportedDescriptions(): HashMap<String, String>
+ hashCode(): int
+ equals(obj: Object): boolean

<<Singleton>>
extensibleclustering:: plugins::PluginExecutor
Runs a given plugin using the specified arguments.

Responsibilities:
+ execParser (file: Path, parser: Class<?>): Position[]
+ execClusteringAlgorithm(data: Position[], clusteringAlgo: Class<?>): Path
+ execVisualisationMethod(file: Path, visualisationMethod: Class<?>): Scene

---

<<Interface>>
extensibleclustering:: plugins::PluginType
Interface to be implemented by each of the PluginTypes. These will modify the correct section of the ImportedPlugins Maps.

Responsibilities:
+ storePlugin (importedPlugins: ImportedPlugins, name: String, plugin: Class<?>)

<<Factory>>
extensibleclustering:: plugins::PluginTypeFactory
Factory that gets the correct PluginType class depending on the class given

Attributes:
- parserType: ParserType

Responsibilities:
<<constructor>> + PluginTypeFactory()
+ getPluginType(interfaceName: String): PluginType

---

extensibleclustering:: plugins::PluginLoader
Class that is responsible for loading all of the plugins from the plugin directory.

Responsibilities:
+ loadPlugins(): ImportedPlugins

<<Singleton>>
extensibleclustering:: plugins::ParserType
Modifies the parsers stored in the ImportedPlugin parser map.

Responsibilities:
+ storePlugin (importedPlugins: ImportedPlugins, name: String, plugin: Class<?>)

<<Singleton>>
extensibleclustering:: plugins::ClusteringType
Modifies the clustering algorithms stored in the ImportedPlugin clustering map.

Responsibilities:
+ storePlugin (importedPlugins: ImportedPlugins, name: String, plugin: Class<?>)

<<Singleton>>
extensibleclustering:: plugins::VisualisationType
Modifies the visualisation algorithms stored in the ImportedPlugin visualisation map.

Responsibilities:
+ storePlugin (importedPlugins: ImportedPlugins, name: String, plugin: Class<?>)

Uses