Package es.uco.kdis.isbse

Interface Summary

IInteractiveAlgorithm

An interface that specifies the operations that every interactive SBSE algorithm should implement.

Class Summary

InteractionContext

A context class to maintain information of the application in which the interactive sessions are happening.

Language

Enumeration of possibles languages.

---

es.uco.kdis.isbse

Interface IInteractiveAlgorithm

< Methods >

---

public interface **IInteractiveAlgorithm**

An interface that specifies the operations that every interactive SBSE algorithm should implement. At the moment, it only requires access to an interactive session, which will contain all the necessary information to manage user's interaction. Each interactive session comprises a series of interactive events. Notice that concurrent access to the interactive session might need to be guaranteed, but it is an implementation detail that will rely on the specific algorithm implementation.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

InteractiveSession

InteractiveEvent

Methods

getInteractionContext

`public InteractionContext getInteractionContext()`

Get access to the interaction context.

Returns:

Current interaction context.

---

getInteractiveSession

`public InteractiveSession getInteractiveSession()`

Get access to the interactive session.

Returns:

Current interactive session.

---

setInteractionContext

`public void setInteractionContext(InteractionContext context)`

Set the interactive context.

Parameters:

context - New interaction context.

---

setInteractiveSession

`public void setInteractiveSession(InteractiveSession session)`

Set the interactive session.

Parameters:

session - New interactive session.

es.uco.kdis.isbse

Class InteractionContext

```
java.lang.Object
    |
    +--es.uco.kdis.isbse.InteractionContext
```

< Constructors > < Methods >

public class **InteractionContext**
extends java.lang.Object

A context class to maintain information of the application in which the interactive sessions are happening. It is expected that this API can be used to support the development of different interactive algorithms and applications, each one having their own characteristics and goals. In order to be able to adapt the interactive experience to a particular context, it might be necessary to know some of these characteristics, such as the language used to show information to the user. Therefore, this class is intended to provide a way to share this information.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IInteractiveAlgorithm

Language

InteractiveSession

InteractiveEvent

Constructors

InteractionContext

`public` **`InteractionContext`**`()`

Empty constructor.

Methods

getIdentifier

`public java.lang.String` **`getIdentifier`**`()`

Get access to the identifier.

Returns:

The identifier as a string.

---

getLanguage

`public` `Language` **`getLanguage`**`()`

Get access to the language.

Returns:

The 2-letter acronym of the language.

---

getNumberOfEvents

`public int` **`getNumberOfEvents`**`()`

Get the number of events comprising each session.

Returns:

The number of events.

---

getNumberOfSessions

`public int` **`getNumberOfSessions`**`()`

Get the number of sessions comprising the interaction.

Returns:

The number of sessions.

---

setIdentifier

`public void **setIdentifier**(java.lang.String id)`

Set the value of the identifier.

Parameters:

id - New value.

---

setLanguage

`public void **setLanguage**(Language language)`

Set the language.

Parameters:

language - The language.

---

setNumberOfEvents

`public void **setNumberOfEvents**(int numberOfEvents)`

Set the number of events.

Parameters:

numberOfEvents - Number of events.

---

setNumberOfSessions

`public void **setNumberOfSessions**(int numberOfSessions)`

Set the number of sessions.

Parameters:

numberOfSessions - Number of sessions.

es.uco.kdis.isbse

Class Language

```
java.lang.Object
    |
    +--java.lang.Enum
        |
        +--es.uco.kdis.isbse.Language
```

All Implemented Interfaces:
java.io.Serializable, java.lang.Comparable

< Fields > < Methods >

---

public final class **Language**
extends java.lang.Enum

Enumeration of possibles languages.

- EN: English
- ES: Spanish
- FR: French
- GE: German
- IT: Italian
- JA: Japanese
- PT: Portuguese
- ZH: Chinese

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

InteractiveEvent

Fields

EN

```
public static final Language EN
```

ES

public static final Language **ES**

FR

public static final Language **FR**

GE

public static final Language **GE**

IT

public static final Language **IT**

JA

public static final Language **JA**

PT

public static final Language **PT**

ZH

public static final Language **ZH**

Methods

valueOf

public static Language **valueOf**(java.lang.String name)

values

```
public static es.uco.kdis.isbse.Language[] values()
```

Package es.uco.kdis.isbse.action

Interface Summary

ICompareSolutions

An interface that specifies the operations that allow the user to compare two or more solutions.

IFreezeSolution

An interface that specifies the operations to allow the user to freeze a solution or a part of it.

IModifySolution

An interface that specifies the operations that allow the user to modify a solution.

IMultipleUserAction

An interface that specifies the operations that an interactive algorithm has to implement to allow the user to perform multiple actions during the interaction.

IRemoveSolution

An interface that specifies the operations that allow the user to remove a solution.

ISaveSolution

An interface that specifies the operations that allow the user to save a solution.

ISelectSolution

An interface that specifies the operations that allow the user to select a solution.

IUserAction

An interface that specifies the operations that the interactive algorithm has to implement to allow the user to perform a particular action during the interaction.

Class Summary

InteractiveActionType

Enumeration of possible types of actions.

---

es.uco.kdis.isbse.action

Interface ICompareSolutions

All Implemented Interfaces:
IUserAction

< Methods >

---

public interface **ICompareSolutions**
implements IUserAction

An interface that specifies the operations that allow the user to compare two or more solutions. Depending on the designed interactive event, the user can indicate which solution is the best/worst or

specify a order for each one.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IUserAction

Methods

getBest

`public int **getBest**()`

Get the best solution

Returns:

The index of the best solution, -1 if it has not been assigned yet.

---

getNumberOfSolutions

`public int **getNumberOfSolutions**()`

Get the number of solutions presented to the user for comparison.

Returns:

Number of solutions.

---

getPosition

```
public int getPosition(int index)
```

Get the position of a specific solution.

Parameters:

index - The index of the solution.

Returns:

The position in the comparison for the solution, -1 if it has not been assigned yet or the index is out of bounds.

---

getWorst

```
public int getWorst()
```

Get the worst solution.

Returns:

The index of the worst solution, -1 if it has not been assigned yet.

---

setBest

```
public void setBest(int index)
```

Set the best solution.

Parameters:

index - The index of the solution.

---

setPosition

```
public void setPosition(int index,
                        int position)
```

Set the position for a specific solution.

Parameters:

index - The index of the solution.
position - The position in the comparison.

---

setWorst

`public void setWorst(int index)`

Set the worst solution.

Parameters:

index - The index of the solution.

---

es.uco.kdis.isbse.action

Interface IFreezeSolution

All Implemented Interfaces:
IModifySolution

< Methods >

---

public interface **IFreezeSolution**
implements IModifySolution

An interface that specifies the operations to allow the user to freeze a solution or a part of it. With this action, the user can specify which part of the solution should not be modify by the search algorithm. For the sake of generality, the interface provides operations to completely freeze the solution or just an element of it, which should be identified by an index. Specific implementations will decide how a solution can be split and how many elements can simultaneously be frozen.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IModifySolution

Methods

getFrozenElement

`public int` **`getFrozenElement`**`()`

Get the index of the first element that is frozen.

Returns:

Index of the frozen element, -1 if no element is frozen or the complete solution is frozen.

---

getFrozenElements

`public int[]` **`getFrozenElements`**`()`

Get the index of the elements that are frozen.

Returns:

Array with the indexes of the frozen elements, null if no element is frozen or the complete solution is frozen.

---

getNumberFreezableElements

`public int` **`getNumberFreezableElements`**`()`

Get the number of elements that can simultaneously be frozen.

Returns:

Number of elements that can be frozen.

---

getNumberOfElements

`public int` **`getNumberOfElements`**`()`

Get the number of elements comprising the solution.

Returns:

Number of constituting elements.

---

isFrozenSolution

`public boolean` **`isFrozenSolution`**`()`

A method to know whether the current solution has been frozen either completely or partially.

Returns:

True if the solution is frozen, false otherwise.

---

setFrozenElement

`public void **setFrozenElement**(int index)`

Specify the element of the solution that should be frozen.

Parameters:

index - Element index.

---

setFrozenSolution

`public void **setFrozenSolution**(boolean frozen)`

Specify if the solution should be frozen, either particularly or completely.

Parameters:

frozen - New value.

---

es.uco.kdis.isbse.action

Interface IModifySolution

All Implemented Interfaces:
IUserAction

< Methods >

---

public interface **IModifySolution**
implements IUserAction

An interface that specifies the operations that allow the user to modify a solution.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IUserAction

Methods

setUpdatedSolution

`public void **setUpdatedSolution**(IInteractiveSolution solution)`

Set the updated solution

Parameters:

solution - The updated solution

---

es.uco.kdis.isbse.action

Interface IMultipleUserAction

< Methods >

public interface **IMultipleUserAction**

An interface that specifies the operations that an interactive algorithm has to implement to allow the user to perform multiple actions during the interaction. A multiple user action is simply a group of individual actions that are available for the user in the same interactive event. The specific composition of this group depends on the specific interactive algorithm.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:

Aurora Ramirez (AR)

Author:

Jose Raul Romero (JRR)

Author:

Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:

1.0

IUserAction

Methods

clearAllActions

`public void clearAllActions()`

Clear the information related to every action in the multiple action.

---

getUserActionByIndex

`public IUserAction getUserActionByIndex(int index)`

Get the action at a specific position

Parameters:

index - Action position.

Returns:

The action at the given position, null if the index is out of bounds.

---

getUserActionByType

`public IUserAction getUserActionByType(InteractiveActionType type)`

Get the action of a specific type.

Parameters:

type - Type of action.

Returns:

The action having the desired type, null if no action has this type.

---

getUserActions

`public java.util.List getUserActions()`

Get the list of actions.

Returns:

A list with the selected actions.

---

es.uco.kdis.isbse.action

Interface IRemoveSolution

All Implemented Interfaces:

ISelectSolution

< Methods >

public interface **IRemoveSolution**
implements ISelectSolution

An interface that specifies the operations that allow the user to remove a solution. With this action, the user can specify that a solution should not be considered anymore by the search algorithm.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IUserAction

Methods

isSolutionToBeRemoved

`public boolean` **`isSolutionToBeRemoved`**`()`

Specify whether the current solution has been marked to be removed.

Returns:

True if the user wants to remove this solution, false otherwise.

setSolutionToBeRemoved

`public void` **`setSolutionToBeRemoved`**`(boolean remove)`

Set whether the solution should be removed.

Parameters:

remove - New value.

es.uco.kdis.isbse.action

Interface ISaveSolution

All Implemented Interfaces:
ISelectSolution

< Methods >

public interface **ISaveSolution**
implements ISelectSolution

An interface that specifies the operations that allow the user to save a solution. With this action, the user can specify that a solution should be kept by the search algorithm.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IUserAction

Methods

isSolutionToBeSaved

`public boolean` **`isSolutionToBeSaved`**`()`

Specify whether the current solution has been marked to be saved.

Returns:

True if the user wants to save this solution, false otherwise.

setSolutionToBeSaved

`public void` **`setSolutionToBeSaved`**`(boolean save)`

Set whether the solution should be saved.

Parameters:

save - New value.

---

es.uco.kdis.isbse.action

Interface ISelectSolution

All Implemented Interfaces:
IUserAction

< Methods >

---

public interface **ISelectSolution**
implements IUserAction

An interface that specifies the operations that allow the user to select a solution. With this action, the user can specify which solutions he/she prefers.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IUserAction

Methods

getIndexSelectedSolution

`public int getIndexSelectedSolution()`

Get the index of the selected solution.

Returns:

Position index.

---

setIndexSelectedSolution

`public void setIndexSelectedSolution(int index)`

Set the index of the selected solution.

Parameters:

index - Position index.

---

es.uco.kdis.isbse.action

Interface IUserAction

< Methods >

---

public interface **IUserAction**

An interface that specifies the operations that the interactive algorithm has to implement to allow the user to perform a particular action during the interaction. Every action should be characterized by a type. Different types of actions can be performed (e.g. evaluate, compare, select, modify solutions) by extending this interface. Additional operations that are common to any operation are defined in this interface, including access to a description of the action and the possibility to remove any information associated to the user's action.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:

Aurora Ramirez (AR)

Author:

Jose Raul Romero (JRR)

Author:

Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:

1.0

InteractiveActionType

Methods

clearAction

`public void **clearAction**()`

Delete any information associated to this action.

---

getActionDescription

`public java.lang.String **getActionDescription**()`

Get the instructions for the user action.

Returns:

A string with the description.

---

getActionName

`public java.lang.String **getActionName**()`

Get the name of the user action.

Returns:

A string with the name.

---

getActionType

`public InteractiveActionType **getActionType**()`

Get the type of action.

Returns:

The type of the action.

---

setContext

`public void **setContext**(InteractionContext context)`

Set the interaction context.

Parameters:

context - The context.

es.uco.kdis.isbse.action

Class InteractiveActionType

```
java.lang.Object
    |
    +--java.lang.Enum
        |
        +--es.uco.kdis.isbse.action.InteractiveActionType
```

All Implemented Interfaces:
java.io.Serializable, java.lang.Comparable

< Fields > < Methods >

public final class **InteractiveActionType**
extends java.lang.Enum

Enumeration of possible types of actions.

- EVALUATION: The aim of the action is to provide an evaluation.
- SELECTION: The aim of the action is to select a solution.
- COMPARISON: The aim of the action is to compare solutions.
- MODIFICATION: The aim of the action is to modify a solution.
- MULTIPLE: Several types of actions are allowed.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IUserAction

Fields

COMPARISON

`public static final InteractiveActionType` **`COMPARISON`**

EVALUATION

public static final InteractiveActionType **EVALUATION**

MODIFICATION

public static final InteractiveActionType **MODIFICATION**

MULTIPLE

public static final InteractiveActionType **MULTIPLE**

SELECTION

public static final InteractiveActionType **SELECTION**

Methods

valueOf

public static InteractiveActionType **valueOf**(java.lang.String name)

---

values

public static es.uco.kdis.isbse.action.InteractiveActionType[] **values**()

Package es.uco.kdis.isbse.evaluation

Interface Summary

IEvaluateSolution

An interface that specifies the operations to perform the evaluation of a solution.

IFitnessEvaluation

An interface that specifies the operations to evaluate the fitness of a solution.

IRankingEvaluation

An interface that specifies the operations to evaluate a solution using rankings.

IRewardEvaluation

An interface that specifies the operations to evaluate a solution using a reward/penalization approach.

IScoreEvaluation

An interface that specifies the operations to evaluate a solution using scores.

IWeightEvaluation

An interface that specifies the operations to evaluate a solution using weights.

Class Summary

InteractiveEvaluationType

Enumeration of possible types of evaluation mechanisms.

---

es.uco.kdis.isbse.evaluation

Interface IEvaluateSolution

All Implemented Interfaces:
IUserAction

< Methods >

---

public interface **IEvaluateSolution**
implements IUserAction

An interface that specifies the operations to perform the evaluation of a solution. This interface only declares general operations which are common to any kind of evaluation. The interactive algorithm should implement one of the interfaces extending this interface to effectively include an evaluation mechanism (scores, rankings, etc.).

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:

Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IUserAction

InteractiveEvaluationType

Methods

getEvaluationType

`public InteractiveEvaluationType getEvaluationType()`

Get the type of evaluation.

Returns:

The type of evaluation.

---

es.uco.kdis.isbse.evaluation

Interface IFitnessEvaluation

All Implemented Interfaces:
IEvaluateSolution

< Methods >

---

public interface **IFitnessEvaluation**
implements IEvaluateSolution

An interface that specifies the operations to evaluate the fitness of a solution.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:

Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:

1.0

IEvaluateSolution

Methods

getFitnessValue

`public double getFitnessValue()`

Get the fitness value.

Returns:

Fitness value.

---

getLowerBound

`public double getLowerBound()`

Get the minimum possible value.

Returns:

Lower bound of the fitness function.

---

getUpperBound

`public double getUpperBound()`

Get the maximum possible value.

Returns:

Upper bound of the fitness function.

---

setFitnessValue

`public void setFitnessValue(double fitness)`

Set the fitness value.

Parameters:

fitness - New fitness.

es.uco.kdis.isbse.evaluation

Interface IRankingEvaluation

All Implemented Interfaces:
IUserAction

< Methods >

public interface **IRankingEvaluation**
implements IUserAction

An interface that specifies the operations to evaluate a solution using rankings.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IEvaluateSolution

Methods

getNumberRankingPositions

`public int` **`getNumberRankingPositions`**`()`

Get the number of possible ranking positions.

Returns:

An integer with the lowest ranking.

getRanking

```
public int getRanking(int index)
```

Get ranking of solution at a specified position.

Parameters:

index - The position index.

Returns:

The ranking position.

---

setRanking

```
public void setRanking(int ranking,
                       int index)
```

Set ranking for a solution.

Parameters:

ranking - The ranking value.
index - The position index.

---

es.uco.kdis.isbse.evaluation

Interface IRewardEvaluation

All Implemented Interfaces:
IUserAction

< Methods >

---

public interface **IRewardEvaluation**
implements IUserAction

An interface that specifies the operations to evaluate a solution using a reward/penalization approach. The evaluation consists in a set of preferences that determine the characteristics in which the user is interested (or not). Different types of preferences are defined by means of additional interfaces.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IEvaluateSolution

IPreference

Methods

getListOfPreferences

`public java.util.List` **`getListOfPreferences`**`()`

Get the list of preferences that are applicable to the solution.

Returns:

List of preferences.

---

getSelectedPreference

`public IPreference` **`getSelectedPreference`**`()`

Get the preference that the user has selected.

Returns:

Selected preference.

---

getSelectedPreferences

`public java.util.List` **`getSelectedPreferences`**`()`

Get the preferences that the user has selected if multiple preferences are allowed.

Returns:

The list of preferences selected by the user.

---

isMultipleRewardAllowed

`public boolean` **`isMultipleRewardAllowed`**`()`

Get whether more than one preference can be selected in each interaction.

Returns:

True if multiple preferences are allowed, false otherwise.

rewardSolution

`public void rewardSolution(IPreference preference)`

Set the preference that the user has selected.

Parameters:

preference - A preference to be rewarded.

rewardSolution

`public void rewardSolution(java.util.List preferences)`

Set the preferences that the user has selected.

Parameters:

preferences - The list of preferences to be rewarded.

setListOfPreferences

`public void setListOfPreferences(java.util.List preferences)`

Set the list of preferences that are applicable to the solution.

Parameters:

preferences - List of preferences.

es.uco.kdis.isbse.evaluation

Interface IScoreEvaluation

All Implemented Interfaces:
IEvaluateSolution

< Methods >

public interface **IScoreEvaluation**
implements IEvaluateSolution

An interface that specifies the operations to evaluate a solution using scores. Scores are discrete values (probably specific for the problem) that are presented to the user so he/she can choose those that apply to the solution.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:

Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IEvaluateSolution

Methods

deselectScore

`public void **deselectScore**(int index)`

Set that a specific score should be deselected.

Parameters:

index - Index of the score to be deselected.

---

getNumberOfScores

`public int **getNumberOfScores**()`

Get the number of possible values.

Returns:

Number of scores.

---

getScoreNames

`public java.util.List **getScoreNames**()`

Get the name of the possible scores.

Returns:

A list containing the names of the scores.

---

getSelectedScore

`public int **getSelectedScore**()`

Get the index of the selected score.

Returns:

Index of the selected score.

---

getSelectedScores

`public int[] **getSelectedScores**()`

Get the indexes of the selected scores.

Returns:

Indexes of the selected scores.

---

isMultipleOptionAllowed

`public boolean **isMultipleOptionAllowed**()`

Get whether more than one score can be selected.

Returns:

True if multiple options can be selected, false otherwise.

---

isScoreSelected

`public boolean **isScoreSelected**(int index)`

Get whether a specific score has been selected.

Parameters:

index - Index of the score.

Returns:

True if the score at the given position was selected, false otherwise.

---

selectScore

`public void **selectScore**(int index)`

Set that a specific score has been assigned.

Parameters:

index - Index of the selected score.

es.uco.kdis.isbse.evaluation

Interface IWeightEvaluation

All Implemented Interfaces:
IEvaluateSolution

< Methods >

public interface **IWeightEvaluation**
implements IEvaluateSolution

An interface that specifies the operations to evaluate a solution using weights. Weights will usually be used to specify the trade-off among objective functions in multi-objective problems.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IEvaluateSolution

Methods

getMaximumWeight

`public double` **`getMaximumWeight`**`()`

Get the maximum value that can be assigned to a weight.

Returns:

Maximum possible value.

getMinimumWeight

`public double` **`getMinimumWeight`**`()`

Get the minimum value that can be assigned to a weight.

Returns:

Minimum possible value.

---

getNumberOfWeights

`public int` **`getNumberOfWeights`**`()`

Get the number of weights to be assigned.

Returns:

Number of weights.

---

getWeight

`public double` **`getWeight`**`(int index)`

Get the weight for a specific position.

Parameters:

index - Position index.

Returns:

The weight value at the specified position.

---

getWeightNames

`public java.util.List` **`getWeightNames`**`()`

Get the name of the property or function associated to each weight.

Returns:

A list with the names as strings.

---

getWeights

`public double[]` **`getWeights`**`()`

Get the values for all weights.

Returns:

An array with the weights.

setWeight

```
public void setWeight(int index,
                      double weight)
```

Set the weight for a specific position.

Parameters:

index - Position index.
weight - Weight value.

setWeights

```
public void setWeights(double[] weights)
```

Set the values for all weights.

Parameters:

weights - New weights.

es.uco.kdis.isbse.evaluation

Class InteractiveEvaluationType

```
java.lang.Object
    |
    +--java.lang.Enum
        |
        +--es.uco.kdis.isbse.evaluation.InteractiveEvaluationType
```

All Implemented Interfaces:
java.io.Serializable, java.lang.Comparable

< Fields > < Methods >

public final class **InteractiveEvaluationType**
extends java.lang.Enum

Enumeration of possible types of evaluation mechanisms.

- FITNESS: The user is asked to provide a fitness value.
- WEIGHTS: The user is asked to assign a set of weights.
- SCORE: The user is asked to give some scores.
- RANKING: The user is asked to rank a solution.
- REWARD: The user is asked to reward/penalize some characteristics.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IEvaluateSolution

Fields

FITNESS

`public static final InteractiveEvaluationType FITNESS`

RANKING

`public static final InteractiveEvaluationType RANKING`

REWARD

`public static final InteractiveEvaluationType REWARD`

SCORE

`public static final InteractiveEvaluationType SCORE`

WEIGHTS

`public static final InteractiveEvaluationType WEIGHTS`

Methods

valueOf

`public static InteractiveEvaluationType valueOf(java.lang.String name)`

---

values

```
public static es.uco.kdis.isbse.evaluation.InteractiveEvaluationType[]
values()
```

Package es.uco.kdis.isbse.evaluation.preference

Interface Summary

ICharacteristicPreference

An interface that specifies the operations to reward or penalize the presence of a structural characteristic in the solution.

IMetricPreference

An interface that specifies the operations to indicate a preference with respect to a metric.

IMetricRangePreference

An interface that specifies the operations to indicate a preference with respect to the range of a metric.

IMetricValuePreference

An interface that specifies the operations to indicate a preference with respect to the value of a metric.

INoPreference

An interface that specifies the lack of preference.

IPreference

An interface that specifies the operations to manage preferences in reward/penalization evaluation.

IReferencePointPreference

An interface that specifies the operations to indicate a desired point in the objective space.

Class Summary

PreferenceType

Enumeration of possible types of preferences for reward/penalization evaluation.

---

es.uco.kdis.isbse.evaluation.preference

Interface ICharacteristicPreference

All Implemented Interfaces:
IPreference

< Methods >

---

public interface **ICharacteristicPreference**
implements IPreference

An interface that specifies the operations to reward or penalize the presence of a structural characteristic

in the solution.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IPreference

Methods

deselectOption

`public void **deselectOption**(int index)`

Set that a specific option should be deselected.

Parameters:

index - Index of the option to be deselected.

---

getNumberOfOptions

`public int **getNumberOfOptions**()`

Get the number of possible values.

Returns:

Number of options.

---

getOptions

`public java.util.List **getOptions**()`

Get the set of options that can be selected.

Returns:

List of strings representing the possible options.

getSelectedOption

`public int **getSelectedOption**()`

Get the selected option.

Returns:

Index of the selected option, -1 if no option has been selected yet.

getSelectedOptions

`public int[] **getSelectedOptions**()`

Get the selected options.

Returns:

Indexes of the selected options, null if no options have been selected yet.

isMultipleOptionAllowed

`public boolean **isMultipleOptionAllowed**()`

Get whether more than one value can be selected.

Returns:

True if multiple options can be selected, false otherwise.

isOptionSelected

`public boolean **isOptionSelected**(int index)`

Get whether a specific option has been selected

Parameters:

index - Index of the option.

Returns:

True if the option at the given position was selected, false otherwise.

selectOption

`public void **selectOption**(int index)`

Set the selected option.

Parameters:

index - Index of the selected option.

es.uco.kdis.isbse.evaluation.preference

Interface IMetricPreference

< Methods >

public interface **IMetricPreference**

An interface that specifies the operations to indicate a preference with respect to a metric. It defines the common operations for specific types of metric evaluation (range, value), so the interfaces extending from it are the ones to be implemented.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IPreference

Methods

getMetricLowerBound

`public double` **`getMetricLowerBound`**`()`

Get the minimum possible value for the metric.

Returns:

Lower bound of the metric.

getMetricName

`public java.lang.String` **`getMetricName`**`()`

Get the name of the metric.

Returns:

A string with the name of the metric.

---

getMetricUpperBound

`public double` **`getMetricUpperBound`**`()`

Get the maximum possible value for the metric.

Returns:

Upper bound of the metric.

---

es.uco.kdis.isbse.evaluation.preference

Interface IMetricRangePreference

All Implemented Interfaces:
IMetricPreference

< Methods >

---

public interface **IMetricRangePreference**
implements IMetricPreference

An interface that specifies the operations to indicate a preference with respect to the range of a metric.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IPreference

Methods

getRangeMaximum

```
public double getRangeMaximum()
```

Get the value that the user has set as the maximum.

Returns:

Upper bound of the range, Double.NaN if it has not been set yet.

---

getRangeMinimum

```
public double getRangeMinimum()
```

Get the value that the user has set as the minimum.

Returns:

Lower bound of the range, Double.NaN if it has not been set yet.

---

setRange

```
public void setRange(double min,
                     double max)
```

Set the range selected by the user.

Parameters:

min - Minimum value expected.
max - Maximum value expected.

---

es.uco.kdis.isbse.evaluation.preference

Interface IMetricValuePreference

All Implemented Interfaces:
IMetricPreference

< Methods >

---

public interface **IMetricValuePreference**
implements IMetricPreference

An interface that specifies the operations to indicate a preference with respect to the value of a metric.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IPreference

Methods

getPreferredValue

`public double` **`getPreferredValue`**`()`

Get the value preferred by the user.

Returns:

The selected value, Double.NaN if it has not been set yet.

---

setPreferredValue

`public void` **`setPreferredValue`**`(double value)`

Set the value preferred by the user.

Parameters:

value - The selected value.

---

es.uco.kdis.isbse.evaluation.preference

Interface INoPreference

All Implemented Interfaces:
IPreference

---

public interface **INoPreference**
implements IPreference

An interface that specifies the lack of preference.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IPreference

---

es.uco.kdis.isbse.evaluation.preference

Interface IPreference

< Methods >

---

public interface **IPreference**

An interface that specifies the operations to manage preferences in reward/penalization evaluation. This interface only declares general methods that are common to every possible preference. Specific interfaces to manage the information of each type of preference are provided as extensions of this interface.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

PreferenceType

IRewardEvaluation

Methods

clearPreference

`public void **clearPreference**()`

Delete previous information associated to this preference, if any.

---

getConfidence

`public double **getConfidence**()`

Get the confidence of the user.

Returns:

Confidence value.

---

getConfidenceDescription

`public java.lang.String **getConfidenceDescription**()`

Get the description associated to the confidence value.

Returns:

A string with the text to be shown for the confidence description.

---

getConfidenceName

`public java.lang.String **getConfidenceName**()`

Get the name to be shown for the confidence value.

Returns:

A string with the label to be shown for the confidence field.

---

getMaximumConfidenceValue

`public int **getMaximumConfidenceValue**()`

Get the maximum possible value for the confidence.

Returns:

Maximum value as integer.

getMinimumConfidenceValue

`public int` **`getMinimumConfidenceValue`**`()`

Get the minimum possible value for the confidence.

Returns:

Minimum value as integer.

getPreferenceDescription

`public java.lang.String` **`getPreferenceDescription`**`()`

Get a description of the preference.

Returns:

A string with the description of this preference.

getPreferenceName

`public java.lang.String` **`getPreferenceName`**`()`

Get the name of the preference.

Returns:

A representative name for this preference.

getPreferenceType

`public` `PreferenceType` **`getPreferenceType`**`()`

Get the type of preference.

Returns:

Type of preference.

setConfidence

`public void` **`setConfidence`**`(double confidence)`

Set the confidence of the user.

Parameters:

confidence - Confidence value.

setContext

`public void **setContext**(InteractionContext context)`

Set the interaction context.

Parameters:

context - The context.

---

es.uco.kdis.isbse.evaluation.preference

Interface IReferencePointPreference

All Implemented Interfaces:
IPreference

< Methods >

---

public interface **IReferencePointPreference**
implements IPreference

An interface that specifies the operations to indicate a desired point in the objective space. The reference point is represented by a metric value and a weight for each objective.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IPreference

Methods

areValidWeights

`public boolean **areValidWeights**(double[] weights)`

Get whether the combination of weights is valid. For instance, it usually happens that the sum of the weights should be equal to 1.

Parameters:

weights - Selected weights.

Returns:

True if the combination of weights is valid, false otherwise.

---

getDimension

`public int **getDimension**()`

Get the number of coordinates to specify the reference point.

Returns:

The dimension of the objective space.

---

getDimensionLowerBound

`public double **getDimensionLowerBound**(int index)`

Get the minimum value for the metric at a specific dimension.

Parameters:

index - Index of the metric/dimension.

Returns:

Lower bound of the metric at this position.

---

getDimensionName

`public java.lang.String **getDimensionName**(int index)`

Get the name of the metric at a specific dimension.

Parameters:

index - Index of the metric/dimension.

Returns:

Name of the metric at this position.

---

getDimensionNames

`public java.util.List` **`getDimensionNames`**`()`

Get the name of the metric associated to each dimension.

Returns:

A list with the names of the metrics as strings.

---

getDimensionUpperBound

`public double` **`getDimensionUpperBound`**`(int index)`

Get the maximum value for the metric at a specific dimension.

Parameters:

index - Index of the metric/dimension.

Returns:

Upper bound of the metric at this position.

---

getMaximumWeight

`public double` **`getMaximumWeight`**`(int index)`

Get the maximum weight for the metric at a specific dimension.

Parameters:

index - Index of the metric/dimension.

Returns:

Maximum weight of the metric at this position.

---

getMinimiumWeight

`public double` **`getMinimiumWeight`**`(int index)`

Get the minimum weight for the metric at a specific dimension.

Parameters:

index - Index of the metric/dimension.

Returns:

Minimum weight of the metric at this position.

---

getPoint

```
public double[] getPoint()
```

Get the reference point.

Returns:

Value for each metric.

---

getPointName

```
public java.lang.String getPointName()
```

Get the name to be shown for the reference point value.

Returns:

A string with the label to be shown for the point field.

---

getWeightName

```
public java.lang.String getWeightName()
```

Get the name to be shown for the weight values.

Returns:

A string with the label to be shown for the weight field.

---

getWeights

```
public double[] getWeights()
```

Get the weights.

Returns:

Weight for each metric.

---

setPoint

```
public void setPoint(double[] values,
                     double[] weights)
```

Set the reference point.

Parameters:

values - Value for each metric.
weights - Weight for each metric.

es.uco.kdis.isbse.evaluation.preference

Class PreferenceType

```
java.lang.Object
    |
    +--java.lang.Enum
        |
        +--es.uco.kdis.isbse.evaluation.preference.PreferenceType
```

All Implemented Interfaces:
java.io.Serializable, java.lang.Comparable

< Fields > < Methods >

public final class **PreferenceType**
extends java.lang.Enum

Enumeration of possible types of preferences for reward/penalization evaluation.

- POS_CHARACT: The preference represents a desired (positive) characteristic.
- NEG_CHARACT: The preference represents an undesired (negative) characteristic.
- METRIC_VALUE: The preference represents a metric for which a desired value should be assigned.
- METRIC_RANGE: The preference represents a metric for which a range should be specified.
- REF_POINT: The preference represents a set of metric values that should be reached (reference point).
- NO_PREF: Lack of preference.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IPreference

Fields

METRIC_RANGE

```
public static final PreferenceType METRIC_RANGE
```

METRIC_VALUE

public static final PreferenceType **METRIC_VALUE**

NEG_CHARACT

public static final PreferenceType **NEG_CHARACT**

NO_PREF

public static final PreferenceType **NO_PREF**

POS_CHARACT

public static final PreferenceType **POS_CHARACT**

REF_POINT

public static final PreferenceType **REF_POINT**

Methods

valueOf

public static PreferenceType **valueOf**(java.lang.String name)

values

public static es.uco.kdis.isbse.evaluation.preference.PreferenceType[] **values**()

Package es.uco.kdis.isbse.event

Class Summary

EventState

An abstract class to specify the state transitions of an event, as well as other common operations to all possible states.

EventStateFinished

The state of an event that has been completed.

EventStateIdle

The initial state of a created event.

EventStateInteracting

The state of the event while the user is interacting.

EventStateInterrupted

The state that indicates that the event has been interrupted.

InteractiveEvent

An abstract class that represents an interactive event (user interaction) during the interaction between the algorithm and the engineer.

---

es.uco.kdis.isbse.event

Class EventState

```
java.lang.Object
    |
    +--es.uco.kdis.isbse.event.EventState
```

Direct Known Subclasses:
EventStateFinished, EventStateIdle, EventStateInteracting, EventStateInterrupted

---

< Constructors > < Methods >

---

public abstract class **EventState**
extends java.lang.Object

An abstract class to specify the state transitions of an event, as well as other common operations to all possible states. Each state should be then implemented as a subclass, specifying the behavior of the event in each of the following situations: start, stop, resume, finish.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

InteractiveEvent

Constructors

EventState

`public EventState()`

Methods

finish

`public abstract void finish(InteractiveEvent event)`

The event is completely finished.

Parameters:

event - The event.

---

reset

`public abstract void reset(InteractiveEvent event)`

The event is reseted.

Parameters:

event - The event.

---

resume

`public abstract void **resume**(InteractiveEvent event)`

The event is resumed.

Parameters:

event - The event.

---

start

`public abstract void **start**(InteractiveEvent event)`

The event is started.

Parameters:

event - The event.

---

stop

`public abstract void **stop**(InteractiveEvent event)`

The event is stopped probably due to an unexpected interruption.

Parameters:

event - The event.

---

es.uco.kdis.isbse.event

Class EventStateFinished

```
java.lang.Object
    |
    +--EventState
        |
        +--es.uco.kdis.isbse.event.EventStateFinished
```

< Methods >

---

public class **EventStateFinished**
extends EventState

The state of an event that has been completed. This state is the end point, so there is not possibility to move forward to any other state. Therefore, all methods do nothing. The implementation follows the Singleton design pattern.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

EventState

InteractiveEvent

Methods

finish

`public void **finish**(InteractiveEvent event)`

Overrides:
finish in class EventState

---

getInstance

`public static EventState **getInstance**()`

A static method to get an instance of this state.

Returns:
An instance of the 'InProcess' event state.

---

reset

`public void **reset**(InteractiveEvent event)`

Overrides:
reset in class EventState

---

resume

`public void **resume**(InteractiveEvent event)`

Overrides:

resume in class EventState

---

start

`public void **start**(InteractiveEvent event)`

Overrides:

start in class EventState

---

stop

`public void **stop**(InteractiveEvent event)`

Overrides:

stop in class EventState

---

es.uco.kdis.isbse.event

Class EventStateIdle

```
java.lang.Object
    |
    +--EventState
        |
        +--es.uco.kdis.isbse.event.EventStateIdle
```

< Methods >

---

public class **EventStateIdle**
extends EventState

The initial state of a created event. The implementation follows the Singleton design pattern. This state operates as follows:

- start: Reset all time counters, set a new time period and change to 'Interacting'.
- stop: The state does not change.
- resume: The state does not change.
- reset: The state does not change.
- finish: The state changes to 'Finished'.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

EventState

InteractiveEvent

Methods

finish

`public void **finish**(InteractiveEvent event)`

Overrides:
finish in class EventState

---

getInstance

`public static EventState **getInstance**()`

A static method to get an instance of this state.

Returns:
An instance of the 'New' event state.

---

reset

`public void **reset**(InteractiveEvent event)`

Overrides:
reset in class EventState

---

resume

`public void **resume**(InteractiveEvent event)`

Overrides:

resume in class EventState

---

start

`public void **start**(InteractiveEvent event)`

Overrides:

start in class EventState

---

stop

`public void **stop**(InteractiveEvent event)`

Overrides:

stop in class EventState

---

es.uco.kdis.isbse.event

Class EventStateInteracting

```
java.lang.Object
    |
    +--EventState
        |
        +--es.uco.kdis.isbse.event.EventStateInteracting
```

< Methods >

---

public class **EventStateInteracting**
extends EventState

The state of the event while the user is interacting. The implementation follows the Singleton design pattern. This state operates as follows:

- start: The state does not change.
- stop: Update execution time, initialize a new time period and change to 'Interrupted'.
- resume: The state does not change.
- reset: The state does not change.
- finish: Update execution time and change to 'Finished'.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

EventState

InteractiveEvent

Methods

finish

`public void finish(InteractiveEvent event)`

Overrides:
finish in class EventState

---

getInstance

`public static EventState getInstance()`

A static method to get an instance of this state.

Returns:
An instance of the 'InProcess' event state.

---

reset

`public void reset(InteractiveEvent event)`

Overrides:
reset in class EventState

---

resume

`public void **resume**(InteractiveEvent event)`

Overrides:

resume in class EventState

---

start

`public void **start**(InteractiveEvent event)`

Overrides:

start in class EventState

---

stop

`public void **stop**(InteractiveEvent event)`

Overrides:

stop in class EventState

---

es.uco.kdis.isbse.event

Class EventStateInterrupted

```
java.lang.Object
    |
    +--EventState
        |
        +--es.uco.kdis.isbse.event.EventStateInterrupted
```

< Methods >

---

public class **EventStateInterrupted**
extends EventState

The state that indicates that the event has been interrupted. The implementation follows the Singleton design pattern. This state operates as follows:

- start: The state does not change.
- stop: The state does not change.
- resume: The state does not change.
- reset: Clear actions and change to 'Idle'.
- finish: Update interruption time and change to 'Finished'.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

EventState

InteractiveEvent

Methods

finish

`public void **finish**(InteractiveEvent event)`

Overrides:
finish in class EventState

---

getInstance

`public static EventState **getInstance**()`

A static method to get an instance of this state.

Returns:
An instance of the 'Interrupted' event state.

---

reset

`public void **reset**(InteractiveEvent event)`

Overrides:
reset in class EventState

---

resume

`public void **resume**(InteractiveEvent event)`

Overrides:

resume in class EventState

---

start

`public void **start**(InteractiveEvent event)`

Overrides:

start in class EventState

---

stop

`public void **stop**(InteractiveEvent event)`

Overrides:

stop in class EventState

---

es.uco.kdis.isbse.event

Class InteractiveEvent

```
java.lang.Object
    |
    +--es.uco.kdis.isbse.event.InteractiveEvent
```

< Constructors > < Methods >

---

public abstract class **InteractiveEvent**
extends java.lang.Object

An abstract class that represents an interactive event (user interaction) during the interaction between the algorithm and the engineer. The class includes basic properties and methods to control the time spent during interaction, including possible interruptions. This class also declares methods to handle the actions that an external application can perform over an event (through the session): start, stop, interrupt, and resume. The states of the event and their transitions are processed by a state machine following the State design pattern.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:

Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

EventState

InteractiveSession

Constructors

InteractiveEvent

`public` **`InteractiveEvent`**`(InteractionContext context)`

Parameterized constructor.

Parameters:

context - The interaction context.

Methods

finish

`public void` **`finish`**`()`

Stop this event.

---

getDescription

`public abstract java.lang.String` **`getDescription`**`()`

Get the instructions of the interactive event.

Returns:

A textual description of the interaction.

---

getExecutionTime

`public synchronized long **getExecutionTime**()`

Get the time the user has been interacting in this event.

Returns:

Total execution time.

---

getInterruptionTime

`public synchronized long **getInterruptionTime**()`

Get the time the event has been stopped due to interruptions.

Returns:

Total interruption time.

---

hasFinished

`public synchronized boolean **hasFinished**()`

Get whether this event has finished.

Returns:

True if the event has finished, false otherwise.

---

reset

`public void **reset**()`

Reset this event

---

resume

`public void **resume**()`

Resume this event.

---

setInteractionContext

`public void **setInteractionContext**(InteractionContext context)`

Set the interaction context.

Parameters:

context - Interaction context.

---

start

`public void **start**()`

Start this event.

---

stop

`public void **stop**()`

Interrupt this event.

Package es.uco.kdis.isbse.session

Class Summary

InteractiveSession

A generic class that represents an interactive session (iteration) between the algorithm and the engineer.

SessionState

An abstract class to specify the state transitions of a session, as well as other common operations to all possible states.

SessionStateFinished

The state of a session that has been completed.

SessionStateIdle

A class to represent the state of a session being ready to be processed.

SessionStateInteracting

The state of a session while an event is being processed.

SessionStateInterrupted

The state that indicates that the session has been interrupted.

SessionStateUndefined

The initial state of a created session.

---

es.uco.kdis.isbse.session

Class InteractiveSession

```
java.lang.Object
    |
    +--es.uco.kdis.isbse.session.InteractiveSession
```

< Constructors > < Methods >

---

public class **InteractiveSession**
extends java.lang.Object

A generic class that represents an interactive session (iteration) between the algorithm and the engineer. Each session is comprised of a sequence of interactive events and has access to the context in which it has been planned. The class includes methods to access to the current event being processed, as well as a flag to indicate whether the user decided to stop the interaction before all scheduled events were dispatched. The states of the sessions and their transitions are processed by a state machine following the State design pattern.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

SessionState

InteractiveEvent

Constructors

InteractiveSession

```
public  InteractiveSession(InteractionContext context)
```

Parameterized constructor.

Parameters:

context - The interaction context.

InteractiveSession

```
public  InteractiveSession(InteractionContext context,
                           java.util.List events)
```

Parameterized constructor.

Parameters:

context - The interaction context.
events - List of events for this session.

Methods

addEvent

`public void **addEvent**(InteractiveEvent event)`

Add a new event to the list of events.

Parameters:

event - New event.

---

finish

`public void **finish**(boolean userStoppedInteraction)`

Finish this session.

Parameters:

userStoppedInteraction - True if the user stopped the session, false if the application stops the session.

---

getCurrentEventIndex

`public synchronized int **getCurrentEventIndex**()`

Get the index of the event currently being processed.

Returns:

Index of the event, -1 if no event has started yet.

---

getEvent

`public InteractiveEvent **getEvent**(int index)`

Get the event at a specified position.

Parameters:

index - The event index.

Returns:

The interactive event scheduled in the given position, null if the index is out of bounds or the sequence of events is empty.

---

getNumberOfEvents

`public int` **`getNumberOfEvents`**`()`

Get the number of events for this session.

Returns:

Number of events.

---

getTotalInteractionTime

`public long` **`getTotalInteractionTime`**`()`

Get the total time spent in this interaction. Only the time spent in already finished events is considered.

Returns:

Time in milliseconds, 0 if none of the events has been finished yet.

---

getTotalInterruptionTime

`public long` **`getTotalInterruptionTime`**`()`

Get the amount of time the session has been interrupted. Only the interruption time of finished events is considered.

Returns:

Time in milliseconds, 0 if none of the events has been finished yet.

---

hasFinished

`public synchronized boolean` **`hasFinished`**`()`

Get whether the session has finished.

Returns:

True if the session has finished, false otherwise.

---

hasUserStoppedInteraction

`public boolean` **`hasUserStoppedInteraction`**`()`

Get whether the user stopped the interaction.

Returns:

True if the user stopped the interaction, false otherwise.

nextEvent

`public void **nextEvent**()`

Proceed with the next event in the list.

resume

`public void **resume**()`

Resume this session.

start

`public void **start**()`

Start this session.

stop

`public void **stop**()`

Stop the session.

es.uco.kdis.isbse.session

Class SessionState

```
java.lang.Object
    |
    +--es.uco.kdis.isbse.session.SessionState
```

Direct Known Subclasses:
SessionStateFinished, SessionStateIdle, SessionStateInteracting, SessionStateInterrupted, SessionStateUndefined

< Constructors > < Methods >

public abstract class **SessionState**
extends java.lang.Object

An abstract class to specify the state transitions of a session, as well as other common operations to all possible states. Each state should be then implemented as a subclass, specifying the behavior of the session in each of the following situations: addEvent, start, proceedNextEvent, stop, resume, finish.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

SessionState

InteractiveSession

Constructors

SessionState

`public SessionState()`

Methods

addEvent

`public abstract void addEvent(InteractiveSession session)`

Add a new event to the session.

Parameters:

session - The session.

---

finish

`public abstract void finish(InteractiveSession session)`

Finish the session.

Parameters:

session - The session.

---

nextEvent

`public abstract void **nextEvent**(InteractiveSession session)`

Proceed with the next event in the session.

Parameters:

session - The session.

---

resume

`public abstract void **resume**(InteractiveSession session)`

Resume the session.

Parameters:

session - The session.

---

start

`public abstract void **start**(InteractiveSession session)`

Start a new session.

Parameters:

session - The session.

---

stop

`public abstract void **stop**(InteractiveSession session)`

Stop the session.

Parameters:

session - The session.

---

es.uco.kdis.isbse.session

Class SessionStateFinished

```
java.lang.Object
    |
    +--SessionState
        |
        +--es.uco.kdis.isbse.session.SessionStateFinished
```

< Constructors > < Methods >

public class **SessionStateFinished**
extends SessionState

The state of a session that has been completed. This state is the end point, so there is not possibility to move forward to any other state. The implementation follows the Singleton design pattern.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

SessionState

InteractiveSession

Constructors

SessionStateFinished

`public` **`SessionStateFinished`**`()`

Methods

addEvent

`public void` **`addEvent`**`(InteractiveSession session)`

Overrides:
addEvent in class SessionState

finish

`public void **finish**(InteractiveSession session)`

Overrides:

finish in class SessionState

---

getInstance

`public static SessionState **getInstance**()`

A static method to get an instance of this state.

Returns:

An instance of the 'Finished' session state.

---

nextEvent

`public void **nextEvent**(InteractiveSession session)`

Overrides:

nextEvent in class SessionState

---

resume

`public void **resume**(InteractiveSession session)`

Overrides:

resume in class SessionState

---

start

`public void **start**(InteractiveSession session)`

Overrides:

start in class SessionState

---

stop

```
public void stop(InteractiveSession session)
```

Overrides:

stop in class SessionState

---

es.uco.kdis.isbse.session

Class SessionStateIdle

```
java.lang.Object
    |
    +--SessionState
        |
        +--es.uco.kdis.isbse.session.SessionStateIdle
```

< Constructors > < Methods >

---

public class **SessionStateIdle**
extends SessionState

A class to represent the state of a session being ready to be processed. The implementation follows the Singleton design pattern. This state operates as follows:

- addEvent: While new events are added, the state does not change.
- start: The state changes to 'Interacting'.
- nextEvent: The state does not change.
- stop: The state does not change.
- resume: The state does not change.
- finish: The states changes to 'Finished'.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:

Aurora Ramirez (AR)

Author:

Jose Raul Romero (JRR)

Author:

Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:

1.0

SessionState

InteractiveSession

Constructors

SessionStateIdle

`public SessionStateIdle()`

Methods

addEvent

`public void addEvent(InteractiveSession session)`

Overrides:

addEvent in class SessionState

---

finish

`public void finish(InteractiveSession session)`

Overrides:

finish in class SessionState

---

getInstance

`public static SessionState getInstance()`

A static method to get an instance of this state.

Returns:

An instance of the 'Ready' session state.

---

nextEvent

`public void nextEvent(InteractiveSession session)`

Overrides:

nextEvent in class SessionState

---

resume

`public void **resume**(InteractiveSession session)`

Overrides:

resume in class SessionState

---

start

`public void **start**(InteractiveSession session)`

Overrides:

start in class SessionState

---

stop

`public void **stop**(InteractiveSession session)`

Overrides:

stop in class SessionState

---

es.uco.kdis.isbse.session

Class SessionStateInteracting

```
java.lang.Object
    |
    +--SessionState
        |
        +--es.uco.kdis.isbse.session.SessionStateInteracting
```

< Constructors > < Methods >

---

public class **SessionStateInteracting**
extends SessionState

The state of a session while an event is being processed. The implementation follows the Singleton design pattern. This state operates as follows:

- addEvent: The state does not change.
- start: The state does not change.
- nextEvent: The next event is selected. The state changes to 'Finished' if there are not more events to process.
- stop: The state changes to 'Interrupted'.
- resume: The state does not change.
- finish: The state changes to 'Finished'.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

SessionState

InteractiveSession

Constructors

SessionStateInteracting

`public` **`SessionStateInteracting`**`()`

Methods

addEvent

`public void` **`addEvent`**`(InteractiveSession session)`

Overrides:
addEvent in class SessionState

---

finish

`public void` **`finish`**`(InteractiveSession session)`

Overrides:
finish in class SessionState

---

getInstance

`public static SessionState getInstance()`

A static method to get an instance of this state.

Returns:

An instance of the 'InProcess' session state.

---

nextEvent

`public void nextEvent(InteractiveSession session)`

Overrides:

nextEvent in class SessionState

---

resume

`public void resume(InteractiveSession session)`

Overrides:

resume in class SessionState

---

start

`public void start(InteractiveSession session)`

Overrides:

start in class SessionState

---

stop

`public void stop(InteractiveSession session)`

Overrides:

stop in class SessionState

---

es.uco.kdis.isbse.session

Class SessionStateInterrupted

```
java.lang.Object
    |
    +--SessionState
        |
        +--es.uco.kdis.isbse.session.SessionStateInterrupted
```

< Constructors > < Methods >

---

public class **SessionStateInterrupted**
extends SessionState

The state that indicates that the session has been interrupted. The implementation follows the Singleton design pattern. This state operates as follows:

- addEvent: The state does not change.
- start: The state does not change.
- nextEvent: The state does not change.
- stop: The state does not change.
- resume: The current event is resumed and the state changes to 'Interacting'.
- finish: The current event is finished and the state changes to 'Finished'.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

SessionState

InteractiveSession

Constructors

SessionStateInterrupted

```
public  SessionStateInterrupted()
```

Methods

addEvent

`public void **addEvent**(InteractiveSession session)`

Overrides:

addEvent in class SessionState

---

finish

`public void **finish**(InteractiveSession session)`

Overrides:

finish in class SessionState

---

getInstance

`public static SessionState **getInstance**()`

A static method to get an instance of this state.

Returns:

An instance of the 'Interrupted' session state.

---

nextEvent

`public void **nextEvent**(InteractiveSession session)`

Overrides:

nextEvent in class SessionState

---

resume

`public void **resume**(InteractiveSession session)`

Overrides:

resume in class SessionState

---

start

`public void **start**(InteractiveSession session)`

Overrides:

start in class SessionState

---

stop

`public void **stop**(InteractiveSession session)`

Overrides:

stop in class SessionState

---

es.uco.kdis.isbse.session

Class SessionStateUndefined

```
java.lang.Object
    |
    +--SessionState
        |
        +--es.uco.kdis.isbse.session.SessionStateUndefined
```

< Constructors > < Methods >

---

public class **SessionStateUndefined**
extends SessionState

The initial state of a created session. The implementation follows the Singleton design pattern. This state operates as follows:

- addEvent: The state changes to 'Idle'.
- start: The state does not change.
- nextEvent: The state does not change.
- stop: The state does not change.
- resume: The state does not change.
- finish: The state changes to 'Finished'.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:

Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:

1.0

SessionState

InteractiveSession

Constructors

SessionStateUndefined

`public` **`SessionStateUndefined`**`()`

Methods

addEvent

`public void` **`addEvent`**`(InteractiveSession session)`

Overrides:

addEvent in class SessionState

---

finish

`public void` **`finish`**`(InteractiveSession session)`

Overrides:

finish in class SessionState

---

getInstance

`public static SessionState` **`getInstance`**`()`

A static method to get an instance of this state.

Returns:

An instance of the 'New' session state.

---

nextEvent

`public void **nextEvent**(InteractiveSession session)`

Overrides:

nextEvent in class SessionState

---

resume

`public void **resume**(InteractiveSession session)`

Overrides:

resume in class SessionState

---

start

`public void **start**(InteractiveSession session)`

Overrides:

start in class SessionState

---

stop

`public void **stop**(InteractiveSession session)`

Overrides:

stop in class SessionState

Package es.uco.kdis.isbse.solution

Interface Summary

IGraphicalSolution

An interface that declares the specific operations for visual solutions (e.g. diagrams).

IInteractiveSolution

An interface that specifies the general operations to manage a solution that will be shown to the user during the interactive event.

IInteractiveSolutionSet

An interface that provides access to a collection of solutions to be shown during an interactive event.

IQualityInformation

An interface that specifies the operations to retrieve quality information of the solution from the interactive algorithm.

ITextSolution

An interface that specifies the operations to manage a solution that will be presented in a textual form (e.g. code).

IXMISolution

An interface that allows accessing to solutions in XMI format.

Class Summary

InteractiveSolutionType

Enumeration of possible types of solutions.

---

es.uco.kdis.isbse.solution

Interface IGraphicalSolution

All Implemented Interfaces:
IInteractiveSolution

< Methods >

---

public interface **IGraphicalSolution**
implements IInteractiveSolution

An interface that declares the specific operations for visual solutions (e.g. diagrams).

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:

Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

InteractiveSolutionType

Methods

getSolutionPath

`public java.lang.String` **`getSolutionPath`**`()`

Get the path to the solution, if it should be accessed via the file system.

Returns:

A string representing the path to the solution.

---

es.uco.kdis.isbse.solution

Interface IInteractiveSolution

< Methods >

---

public interface **IInteractiveSolution**

An interface that specifies the general operations to manage a solution that will be shown to the user during the interactive event.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:

1.0

InteractiveSolutionType

Methods

getSolutionName

`public java.lang.String` **`getSolutionName`**`()`

Get a representative name for the solution.

Returns:

A string representing the solution name.

---

getSolutionType

`public` `InteractiveSolutionType` **`getSolutionType`**`()`

Get the type of solution.

Returns:

Type of solution.

---

isSolutionModifiable

`public boolean` **`isSolutionModifiable`**`()`

Get whether the solution is modifiable or not.

Returns:

True if the user can modify the solution, false otherwise.

---

es.uco.kdis.isbse.solution

Interface IInteractiveSolutionSet

< Methods >

---

public interface **IInteractiveSolutionSet**

An interface that provides access to a collection of solutions to be shown during an interactive event. There is not special requirements on how the set is managed, provided it implements the interface `Collection`. Solutions comprising this collection should implement the interface `IInteractiveSolution` or one of its subtypes.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IInteractiveSolution

InteractiveSolutionType

Methods

addSolution

```
public boolean addSolution(IInteractiveSolution solution)
```

Add a solution to the collection.

Parameters:

solution - Solution to be added.

Returns:

True if the solution has been added, false otherwise.

---

addSolution

```
public boolean addSolution(IInteractiveSolution solution,
                           int index)
```

Add a solution to the collection at a specific position.

Parameters:

solution - Solution to be added.
index - The position index.

Returns:

True if the solution has been added, false otherwise.

---

getNumberOfSolutions

`public int` **`getNumberOfSolutions`**`()`

Get the number of solutions comprising the collection.

Returns:

Current number of solutions.

---

getSolution

`public` `IInteractiveSolution` **`getSolution`**`(int index)`

Get the solution at a specified position.

Parameters:

index - The position index.

Returns:

Solution at the specified position, `null` if no solution is found.

---

getSolutionType

`public` `InteractiveSolutionType` **`getSolutionType`**`()`

Get the type of the solutions.

Returns:

Type of stored solutions.

---

getSolutions

`public java.util.Collection` **`getSolutions`**`()`

Get all the solutions comprising the collection.

Returns:

Collection containing current solutions.

---

removeSolution

```
public boolean removeSolution(IInteractiveSolution solution)
```

Remove a solution from the collection.

Parameters:

solution - Solution to be removed.

Returns:

True if the solution has been removed, false otherwise.

---

removeSolution

```
public boolean removeSolution(int index)
```

Remove the solution at a specific position from the collection.

Parameters:

index - The position index.

Returns:

True if the solution has been removed, false otherwise.

---

setSolution

```
public void setSolution(IInteractiveSolution solution,
                        int index)
```

Set the solution at the specified position.

Parameters:

solution - The solution to be set.
index - The position index.

---

setSolutions

```
public void setSolutions(java.util.Collection solutions)
```

Set the collection of solutions.

Parameters:

solutions - New collection of solutions.

es.uco.kdis.isbse.solution

Interface IQualityInformation

< Methods >

public interface **IQualityInformation**

An interface that specifies the operations to retrieve quality information of the solution from the interactive algorithm. The quality information refers to a set of metrics and their values. Two types of metrics can be presented. Single metrics are those metrics for which a single value is assigned (e.g. the fitness value), whereas multiple metrics are those comprised of several values (e.g. a code metric for each class of a project or a design metric for each component of an architecture).

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

Methods

getMetricNames

`public java.util.List` **`getMetricNames`**`()`

Get the names of the quality metrics.

Returns:

List with the names of the metrics as strings.

getMultipleMetricElementNames

`public java.util.List` **`getMultipleMetricElementNames`**`(java.lang.String metric)`

Get the names of the elements comprising a multiple metric.

Parameters:

metric - The name of the desired metric.

Returns:

List with the names of elements as strings.

---

getMultipleMetricNames

`public java.util.List` **`getMultipleMetricNames`**`()`

Get the names of multiple metrics.

Returns:

List with the names of multiple metrics as strings.

---

getMultipleValue

`public double[]` **`getMultipleValue`**`(java.lang.String metric)`

Get the values for a multiple metric.

Parameters:

metric - Name of the metric.

Returns:

Values for this metric, null if the metric does not exist.

---

getNumberOfMetrics

`public int` **`getNumberOfMetrics`**`()`

Get the number of quality metrics.

Returns:

Number of quality metrics.

---

getSingleMetricNames

`public java.util.List` **`getSingleMetricNames`**`()`

Get the names of single metrics.

Returns:

List with the names of single metrics as strings.

---

getSingleValue

`public double` **`getSingleValue`**`(java.lang.String metric)`

Get the value for a single metric.

Parameters:

metric - Name of the metric.

Returns:

Value for this metric, Double.NaN if the metric does not exist.

---

isMultipleMetric

`public boolean` **`isMultipleMetric`**`(java.lang.String metric)`

Get whether a given metric is multiple or not.

Parameters:

metric - Name of the metric.

Returns:

True if the metric is multiple, false otherwise.

---

es.uco.kdis.isbse.solution

Interface ITextSolution

All Implemented Interfaces:
IInteractiveSolution

< Methods >

---

public interface **ITextSolution**
implements IInteractiveSolution

An interface that specifies the operations to manage a solution that will be presented in a textual form (e.g. code).

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IGraphicalSolution

Methods

getSolution

`public java.lang.String` **`getSolution`**`()`

Get the solution as a string.

Returns:

A string representing the solution.

---

setSolution

`public void` **`setSolution`**`(java.lang.String solution)`

Set the solution.

Parameters:

solution - The new solution.

---

es.uco.kdis.isbse.solution

Interface IXMISolution

All Implemented Interfaces:
IGraphicalSolution

< Methods >

---

public interface **IXMISolution**
implements IGraphicalSolution

An interface that allows accessing to solutions in XMI format. It currently gives access to a string containing the XMI specification.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

IGraphicalSolution

Methods

getDiagramInXMI

`public java.lang.String` **`getDiagramInXMI`**`()`

Get the XMI of the diagram.

Returns:

A String with the solution in XMI format.

---

setDiagramInXMI

`public void` **`setDiagramInXMI`**`(java.lang.String xmi)`

Set the solution in XMI format.

Parameters:

xmi - The diagram in XMI format.

es.uco.kdis.isbse.solution

Class InteractiveSolutionType

```
java.lang.Object
    |
    +--java.lang.Enum
        |
        +--es.uco.kdis.isbse.solution.InteractiveSolutionType
```

All Implemented Interfaces:
java.io.Serializable, java.lang.Comparable

< Fields > < Methods >

---

public final class **InteractiveSolutionType**
extends java.lang.Enum

Enumeration of possible types of solutions.

- CLASS_DIAGRAM: A class diagram.
- CODE: A piece of code.
- COMP_DIAGRAM: A component diagram.
- PROJECT_PLANNING: A project planning
- REQ_SPEC: A requirements specification.
- TEST_CASE: A test case.

HISTORY:

- (AR|JRR|SV, 1.0, March 2018) Initial version.

Author:
Aurora Ramirez (AR)

Author:
Jose Raul Romero (JRR)

Author:
Sebastian Ventura (SV)

Knowledge Discovery and Intelligent Systems Research Group (KDIS)

Version:
1.0

InteractiveSolutionType

Fields

CLASS_DIAGRAM

`public static final InteractiveSolutionType` **`CLASS_DIAGRAM`**

CODE

public static final InteractiveSolutionType **CODE**

COMP_DIAGRAM

public static final InteractiveSolutionType **COMP_DIAGRAM**

PROJECT_PLANNING

public static final InteractiveSolutionType **PROJECT_PLANNING**

REQ_SPEC

public static final InteractiveSolutionType **REQ_SPEC**

TEST_CASE

public static final InteractiveSolutionType **TEST_CASE**

Methods

valueOf

public static InteractiveSolutionType **valueOf**(java.lang.String name)

---

values

public static es.uco.kdis.isbse.solution.InteractiveSolutionType[] **values**()