```
<<Interface>>
AdvisorInterface
(Implementation Model::juniper-sa-tool::eu.juniper.sa.tool)
<<Property>> +name : String
<<Property>> +description : String
<<Property>> +enabled : boolean
<<Property>> +disabled : boolean
+execute() : Advice []
+execute(monitoringStartTime : Timestamp) : Advice []
+execute(monitoringStartTime : Timestamp, monitoringEndTime : Timestamp) : Advic...
```

```
AdvisorUsingDatabaseAbstract
(Implementation Model::juniper-sa-tool::eu.juniper.sa.tool)
<<Property>> -monitoringDatabaseConnection : Connection
<<Property>> -juniperApplication : JuniperApplication
<<Property>> -enabled : boolean = true
-QUERY_FIRST_TIMESTAMP : String = "SELECT MIN(time) FROM records;"
-QUERY_LAST_TIMESTAMP : String = "SELECT MAX(time) FROM records;"
+AdvisorUsingDatabaseAbstract(juniperApplication : JuniperApplication, monitoringDatabaseConnection : Connection)
+newInstance(advisorClass : Class<?>, juniperApplication : JuniperApplication, monitoringDatabaseConnection : Connection) : AdvisorInte...
+execute() : Advice []
+execute(monitoringStartTime : Timestamp) : Advice []
#generateFromWhereFragment(metricType : String, metricNames : String []) : String
+isDisabled() : boolean
+setDisabled(disabled : boolean) : void
+setObjectProperties(properties : Properties) : void
-getTimestampOfFirstRecord() : Timestamp
-getTimestampOfLastRecord() : Timestamp
#generateFromWhereFragment(metricType : String, metricNames : String [], useLeftOuterJoins : boolean) : String
```

```
AdvisorDataTransferOverhead
(Implementation Model::juniper-sa-tool::eu.juniper.sa.tool::plugins)
-ADVISOR_NAME : String = AdvisorDataTransferOverhead.class.getSimpleName()
-ADVISOR_DESCRIPTION : String = "This advisor detects Juniper programs with long data"
          + " communication times (i.e., a time spent on waiting for receiving data)"
          + " and short computation. This indicate a very simple program with a high"
          + " data transfer overhead (it should be merged with other programs) or a program"
          + " waiting for data most the time (data flows/stream should be optimized"
          + " in the Juniper application of the Juniper program)."
-ADVICE_NAME : String = "DataTransferOverhead"
-ADVICE_TEXT : String = "The $ running at $"
          + " was receiving data in %f seconds of %f seconds of its total execution time"
          + " (averages are %f seconds for %d receives of data and %f seconds for %d executions)."
          + " That makes %f percentage of execution time spent by receiving data"
          + " (the recommended maximum is %f percentage)."
-QUERY_metricsInProgramRuntime : String[] = {"ProgramGlobalRank", "ProgramDuration"}
-QUERY_metricsInSendReceive : String[] = {"ReceiverGlobalRank", "SendReceiveDuration"}
-QUERY : String = "SELECT ProgramGlobalRank,\n"
          + "  ProgramDurationSum,\n"
          + "  ProgramDurationAvg,\n"
          + "  ProgramDurationCount,\n"
          + "  SendReceiveDurationSum,\n"
          + "  SendReceiveDurationAvg,\n"
          + "  SendReceiveDurationCount,\n"
          + "  SendReceiveDurationSum/ProgramDurationSum AS TransferToExecutionDurationRatio\n"
          + "FROM\n"
          + "  (SELECT m0.numericvalue AS ProgramGlobalRank,\n"
          + "    SUM(m1.numericvalue) AS ProgramDurationSum,\n"
          + "    AVG(m1.numericvalue) AS ProgramDurationAvg,\n"
          + "    COUNT(m1.numericvalue) AS ProgramDurationCount\n"
          + AdvisorUsingDatabaseAbstract.generateFromWhereFragment("ProgramRuntime", QUERY_metricsInProgramRun...
          + "  AND (records.time BETWEEN ? AND ?)\n"
          + "  GROUP BY ProgramGlobalRank\n"
          + "  ) ProgramRuntime\n"
          + "JOIN\n"
          + "  (SELECT m0.numericvalue AS ReceiverGlobalRank,\n"
          + "    SUM(m1.numericvalue) AS SendReceiveDurationSum,\n"
          + "    AVG(m1.numericvalue) AS SendReceiveDurationAvg,\n"
          + "    COUNT(m1.numericvalue) AS SendReceiveDurationCount\n"
          + AdvisorUsingDatabaseAbstract.generateFromWhereFragment("SendReceive", QUERY_metricsInSendReceive)
          + "  AND (records.time BETWEEN ? AND ?)\n"
          + "  GROUP BY ReceiverGlobalRank\n"
          + "  ) SendReceive ON (ProgramRuntime.ProgramGlobalRank = SendReceive.ReceiverGlobalRank)\n"
          + "WHERE SendReceiveDurationSum/ProgramDurationSum     >= ?\n"
          + "ORDER BY TransferToExecutionDurationRatio DESC;"
<<Property>> #receivingToExecutionDurationRatio : double = 0
+getName() : String
+getDescription() : String
+execute(monitoringStartTime : Timestamp, monitoringEndTime : Timestamp) : Advice []
+AdvisorDataTransferOverhead(juniperApplication : JuniperApplication, monitoringDatabaseConnection : Connection)
+main(args : String []) : void
```

```
AdvisorOutOfMemoryPrediction
(Implementation Model::juniper-sa-tool::eu.juniper.sa.tool::plugins)
-ADVISOR_NAME : String = AdvisorOutOfMemoryPrediction.class.getSimpleName()
-ADVISOR_DESCRIPTION : String = "This advisor detect Juniper programs"
          + " where the memory usage is growing over the time by detecting"
          + " a linear trend in memory usage (the linear regression analysis)."
          + " This may indicate potential memory leaks and eventually result"
          + " into performance related issues and OutOfMemoryError errors in a Juniper programs."
-ADVICE_NAME_HEAPMEM : String = "OutOfMemoryPrediction_HeapMemory"
-ADVICE_NAME_NONHEAPMEM : String = "OutOfMemoryPrediction_NonHeapMemory"
-ADVICE_NAME_SWAPFILE : String = "OutOfMemoryPrediction_SwapFile"
-ADVICE_TEXT_MODEL : String = ". The sample linear regression model is Y_{size_in_bytes} = %f + %f * X_{time_in_sec}."
-ADVICE_TEXT_HEAPMEM : String = "The $ running at $"
          + " has the memory usage growing over the time by the approximate rate"
          + " of change %f Bytes per second for the heap memory size"
          + " (the recommended maximum is %f Bytes per second)."
          + " This may result into OutOfMemoryError errors in the program"
-ADVICE_TEXT_HEAPMEM_OMM : String = " on %s (that is %s since the beginning of analyzed data on %s; the heap memory is limited to %d Bytes)"
-ADVICE_TEXT_NONHEAPMEM : String = "The $ running at $"
          + " has the memory usage growing over the time by the approximate rate"
          + " of change %f Bytes per second for the non-heap memory size"
          + " (the recommended maximum is %f Bytes per second)."
          + " This may result into OutOfMemoryError errors in the program"
-ADVICE_TEXT_NONHEAPMEM_OMM : String = " on %s (that is %s since the beginning of analyzed data on %s; the non-heap memory is limited to %d Bytes)"
-ADVICE_TEXT_SWAPSPACE : String = "The $ running at $"
          + " has the memory usage growing over the time by the approximate rate"
          + " of change %f Bytes per second for the swap space size"
          + " (the recommended maximum is %f Bytes per second)."
          + " This can result in performance related issues for the program"
-ADVICE_TEXT_SWAPSPACE_OMM : String = " on %s (that is %s since the beginning of analyzed data on %s; the swap space size is limited to %d Bytes)"
-QUERY_metricsInProgramRuntime : String[] = {"ProgramGlobalRank", "UsedHeapMemory", "UsedNonHeapMemory", "UsedSwapSpaceSize"}
-QUERY : String = "SELECT ProgramRuntime.ProgramGlobalRank AS ProgramGlobalRank, ProgramRuntimeAvg.AvgTime AS AvgTime,\n"
          + "  CASEWHEN(SUM(ProgramRuntime.Time-ProgramRuntimeAvg.AvgTime) = 0, 0, SUM((ProgramRuntime.Time-ProgramRuntimeAvg.AvgTime)*(ProgramRuntime.UsedHeapMemory-ProgramRuntimeAvg.AvgUsedHeapMemory))/SUM(POWER(ProgramRuntime.Time-ProgramRuntimeAvg.AvgTime, 2))) AS Beta1UsedHeapMemory,\n"
          + "  ProgramRuntimeAvg.AvgUsedHeapMemory AS AvgUsedHeapMemory,\n"
          + "  CASEWHEN(SUM(ProgramRuntime.Time-ProgramRuntimeAvg.AvgTime) = 0, 0, SUM((ProgramRuntime.Time-ProgramRuntimeAvg.AvgTime)*(ProgramRuntime.UsedNonHeapMemory-ProgramRuntimeAvg.AvgUsedNonHeapMemory))/SUM(POWER(ProgramRuntime.Time-ProgramRuntimeAvg.AvgTime, 2))) AS Beta1UsedNonHeapMem...
          + "  ProgramRuntimeAvg.AvgUsedNonHeapMemory AS AvgUsedNonHeapMemory,\n"
          + "  CASEWHEN(SUM(ProgramRuntime.Time-ProgramRuntimeAvg.AvgTime) = 0, 0, SUM((ProgramRuntime.Time-ProgramRuntimeAvg.AvgTime)*(ProgramRuntime.UsedSwapSpaceSize-ProgramRuntimeAvg.AvgUsedSwapSpaceSize))/SUM(POWER(ProgramRuntime.Time-ProgramRuntimeAvg.AvgTime, 2))) AS Beta1UsedSwapSpaceSize,...
          + "  ProgramRuntimeAvg.AvgUsedSwapSpaceSize AS AvgUsedSwapSpaceSize\n"
          + "FROM\n"
          + "  (SELECT m0.numericvalue AS ProgramGlobalRank,\n"
          + "    m1.numericvalue AS UsedHeapMemory,\n"
          + "    m2.numericvalue AS UsedNonHeapMemory,\n"
          + "    m3.numericvalue AS UsedSwapSpaceSize,\n"
          + "    DATEDIFF('SECOND', ?, records.time) AS Time\n"
          + AdvisorUsingDatabaseAbstract.generateFromWhereFragment("ProgramRuntime", QUERY_metricsInProgramRuntime)
          + "  AND (records.time BETWEEN ? AND ?)\n"
          + "  ) ProgramRuntime\n"
          + "JOIN\n"
          + "  (SELECT m0.numericvalue AS ProgramGlobalRank,\n"
          + "    AVG(m1.numericvalue) AS AvgUsedHeapMemory,\n"
          + "    AVG(m2.numericvalue) AS AvgUsedNonHeapMemory,\n"
          + "    AVG(m3.numericvalue) AS AvgUsedSwapSpaceSize,\n"
          + "    AVG(DATEDIFF('SECOND', ?, records.time)) AS AvgTime\n"
          + AdvisorUsingDatabaseAbstract.generateFromWhereFragment("ProgramRuntime", QUERY_metricsInProgramRuntime)
          + "  AND (records.time BETWEEN ? AND ?)\n"
          + "  GROUP BY ProgramGlobalRank\n"
          + "  ) ProgramRuntimeAvg ON (ProgramRuntime.ProgramGlobalRank = ProgramRuntimeAvg.ProgramGlobalRank)\n"
          + "GROUP BY ProgramRuntime.ProgramGlobalRank\n"
          + "HAVING Beta1UsedHeapMemory >= ?\n"
          + "OR Beta1UsedNonHeapMemory >= ?\n"
          + "OR Beta1UsedSwapSpaceSize >= ?;"
-QUERY_metricsInProgramRuntime_max : String[] = {"ProgramGlobalRank"}
-QUERY_MAX : String = "SELECT"
          + "  (SELECT numericvalue FROM metrics WHERE name = 'MaxSwapSpaceSize' AND recordid = records.id) AS MaxSwapSpaceSize,\n"
          + "  (SELECT numericvalue FROM metrics WHERE name = 'MaxHeapMemory' AND recordid = records.id) AS MaxHeapMemory,\n"
          + "  (SELECT numericvalue FROM metrics WHERE name = 'MaxNonHeapMemory' AND recordid = records.id) AS MaxNonHeapMemory\n"
          + AdvisorUsingDatabaseAbstract.generateFromWhereFragment("ProgramRuntime", QUERY_metricsInProgramRuntime_max)
          + "  AND (m0.numericvalue = ?) AND (records.time BETWEEN ? AND ?)\n"
          + "LIMIT 1;"
<<Property>> #linearRegressionBeta1ForHeapMemory : double = 0.1
<<Property>> #linearRegressionBeta1ForNonHeapMemory : double = 0.1
<<Property>> #linearRegressionBeta1ForSwapSpace : double = 0.1
+getName() : String
+getDescription() : String
+execute(monitoringStartTime : Timestamp, monitoringEndTime : Timestamp) : Advice []
-getAdviceString(beta1UsedMemory : double, beta1Recommended : double, avgTime : double, avgUsedMemory : double, maxMemory : long, monitoringStartTime : Timestamp, textFirst : String, textOMM : String, textModel : String) : String
-getMilisecondsIntervalBreakdown(miliseconds : long) : String
+AdvisorOutOfMemoryPrediction(juniperApplication : JuniperApplication, monitoringDatabaseConnection : Connection)
+main(args : String []) : void
```

```
Advice
(Implementation Model::juniper-sa-tool::eu.juniper.sa.tool)
<<Property>> -name : String
<<Property>> -problemDescriptionFormat : String
-modelEntities : ModelEntity[]
<<Property>> -solutionDescription : String = null
<<Property>> -noteDescription : String = null
+problemDescriptionFormatEntityMark : char = '$'
+namespacePrefix : String = XMLDeploymentPlan.SAOBJID.getPrefix()
+namespaceURI : String = XMLDeploymentPlan.SAOBJID.getNamespaceURI()
+xMimeNamespacePrefix : String = "xmime"
+xMimeNamespaceURI : String = "http://www.w3.org/2005/05/xmlmime"
+xsiNamespacePrefix : String = "xsi"
+xsiNamespaceURI : String = "http://www.w3.org/2001/XMLSchema-instance"
+schemaLocation : String = "schemaLocation"
+schemaLocationVal : String = "http://www.fit.vutbr.cz/homes/rychly/juniper-sa/xsd/scheduling-advisor-v4.xsd"
-SCHEDULINGADVICE : String = "schedulingAdvice"
-ADVICE : String = "advice"
-CATEGORY : String = "category"
-CATEGORYVAL : String = "resource"
-SEVERITY : String = "severity"
-SEVERITYVAL : String = "warning"
-PROBLEM : String = "problem"
-SOLUTION : String = "solution"
-NOTE : String = "note"
-OBJECTREF : String = "objectRef"
-OBJID : String = "objId"
-SOURCES : String = "sources"
-ATTACHMENTREF : String = "attachmentRef"
-ATTID : String = "attId"
-ATTIDVAL : String = "deployment-plan"
-ATTACHMENT : String = "attachment"
-ANYXML : String = "anyXml"
-CONTENTTYPE : String = "contentType"
-CONTENTTYPEVAL : String = "application/xml"
+toString() : String
+Advice(name : String, problemDescriptionFormat : String, modelEntities : ModelEntity ...)
+getProblemDescriptionAsText() : String
+writeProblemDescriptionIntoXMLStream(xmlStreamWriter : XMLStreamWriter) : void
+writeToXmlStream(xmlStreamWriter : XMLStreamWriter) : void
+writeAdviceArray(adviceArray : Advice [], outputFile : String, juniperApplication : JuniperApplication) : void
+writeAdviceArray(adviceArray : Advice [], outputStream : OutputStream, juniperApplication : JuniperApplication) : void
+writeAdviceArray(adviceArray : Advice [], xmlStreamWriter : XMLStreamWriter, juniperApplication : JuniperApplication) : void
```

```
AdvisorGarbageCollectionPerformance
(Implementation Model::juniper-sa-tool::eu.juniper.sa.tool::plugins)
-ADVISOR_NAME : String = AdvisorGarbageCollectionPerformance.class.getSimpleName()
-ADVISOR_DESCRIPTION : String = "This advisor detects Juniper programs that spent much time on garbage collecting."
          + " Long garbage collections may affect negatively responsiveness of"
          + " a Juniper application which is critical in real-time stream"
          + " processing of Big Data (any delay in processing of such data"
          + " may result into data-loss issues)."
-ADVICE_NAME : String = "GarbageCollectionDelays"
-ADVICE_TEXT : String = "The Java Hotspot JVM of the $ running at $"
          + " was performed %d garbage collections that took %f seconds"
          + " in %f seconds of total execution time of the program"
          + " (averages are %f seconds per garbage collection and %f seconds for the execution time)."
          + " That makes %f percentage of execution time spent by garbage collections"
          + " (the recommended maximum is %f percentage)."
-QUERY_metricsInProgramRuntime : String[] = {
      "ProgramGlobalRank",
      "ProgramDuration",
      "GarbageCollectionCount",
      "GarbageCollectionTime"
   }
-QUERY : String = "SELECT m0.numericvalue AS ProgramGlobalRank,\n"
          + "  SUM(m1.numericvalue) AS ProgramDurationSum,\n"
          + "  AVG(m1.numericvalue) AS ProgramDurationAvg,\n"
          + "  SUM(m2.numericvalue) AS GarbageCollectionCount,\n"
          + "  SUM(m3.numericvalue) AS GarbageCollectionTimeSum,\n"
          + "  CASEWHEN(SUM(m2.numericvalue) = 0, 0, SUM(m3.numericvalue)/SUM(m2.numericvalue)) AS GarbageCollectionTimeAvg,\n"
          + "  CASEWHEN(SUM(m1.numericvalue) = 0, 0, SUM(m3.numericvalue)/SUM(m1.numericvalue)) AS GarbageCollectionToExecutionDurationRat...
          + AdvisorUsingDatabaseAbstract.generateFromWhereFragment("ProgramRuntime", QUERY_metricsInProgramRuntime)
          + "AND (records.time BETWEEN ? AND ?)\n"
          + "GROUP BY ProgramGlobalRank\n"
          + "HAVING GarbageCollectionToExecutionDurationRatio >= ?\n"
          + "ORDER BY GarbageCollectionToExecutionDurationRatio DESC;"
<<Property>> #garbageCollectionToExecutionDurationRatio : double = 0
+getName() : String
+getDescription() : String
+execute(monitoringStartTime : Timestamp, monitoringEndTime : Timestamp) : Advice []
+AdvisorGarbageCollectionPerformance(juniperApplication : JuniperApplication, monitoringDatabaseConnection : Connection)
+main(args : String []) : void
```

```
ClassFinder
(Implementation Model::juniper-sa-tool::eu.juniper.sa.tool::utils)
-DOT : char = '.'
-SLASH : char = '/'
-CLASS_SUFFIX : String = ".class"
-JAR_PREFIX : String = "jar:"
-processDirectory(directory : File, packageName : String, classes : ArrayList<Class<?>>) : void
-processJar(resource : URL, relativePath : String, classes : ArrayList<Class<?>>) : void
+getClassesForPackage(packageName : String) : Class<?> []
+main(args : String []) : void
+getPropertyDescriptors(classWithProperties : Class<?>) : PropertyDescriptor []
+setProperty(objectWithProperties : Object, propertyName : String, propertyValue : String) : boo...
```

```
AdvisorException
(Implementation Model::juniper-sa-tool::eu.juniper.sa.tool)
-serialVersionUID : long = 42L
+AdvisorException(message : String)
+AdvisorException(message : String, cause : Throwable)
+AdvisorException(cause : Throwable)
```

```
Advisor
(Implementation Model::juniper-sa-tool::eu.juniper.sa.tool)
-PLUGINS_PACKAGE : String = "eu.juniper.sa.tool.plugins"
+main(args : String []) : void
```

```
<<enumeration>>
SecondArgType
(Implementation Model::juniper-sa-tool::eu.juniper.sa.tool::Advisor)
MONITORING_SERVICE_URL
SQL_DUMP_FILEPATH
JDBC_TO_MONITORING_DB
```