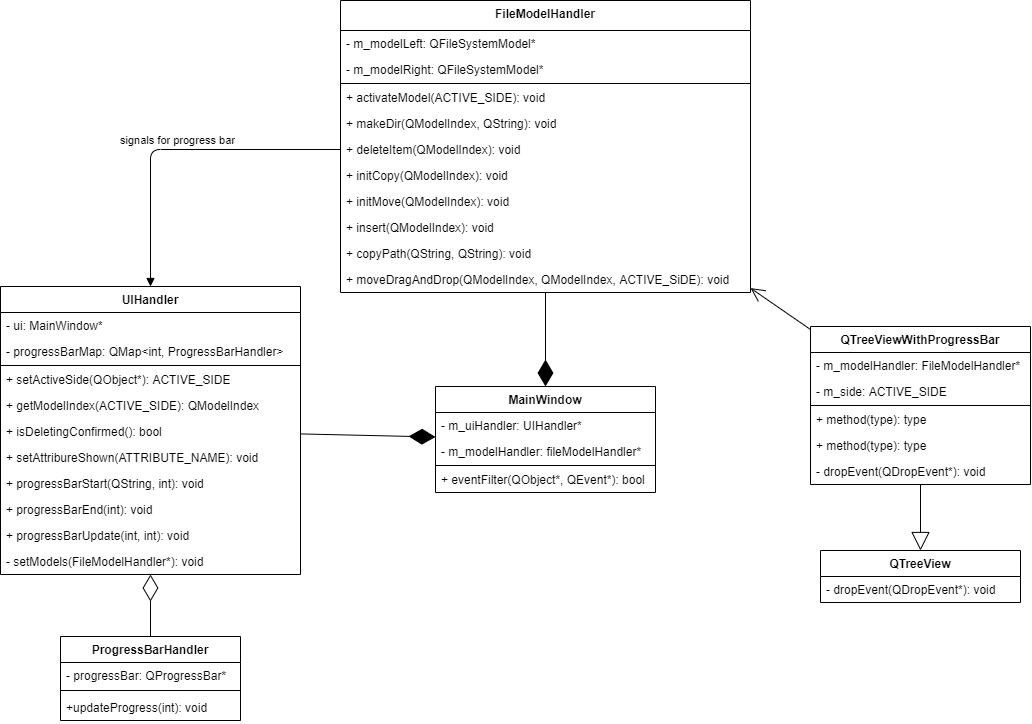
**1. UML Class Diagram**



**2. Object description.**

**2.1 Objects**

**2.1.1 FileModelHandler**

|  |  |
| --- | --- |
| **Class name:** FileModelHandler | |
| **Brief description:** this class is responsible for controlling two QFileSystemModels that provide a data model for the local filesystem. | |
| **Attributes(fields):** | **Attribute description:** |
| QFileSystemModel \*m\_modelLeft | Represents data model for the local filesystem, which is shown at the left window. |
| **Program description language:** |
| private QFileSystemModel \*m\_modelLeft; |
| QFileSystemModel \*m\_modelRight | **Attribute description:** |
| Represents data model for the local filesystem, which is shown at the right window. |
| **Program description language:** |
| private QFileSystemModel \*m\_modelRight; |
| ACTIVE\_SIDE m\_activeSide | **Attribute description:** |
| Represents current active side, the window (left or right) which was clicked by user for the last time. |
| **Program description language:** |
| private ACTIVE\_SIDE m\_activeSide; |
| ACTION\_TYPE m\_currActionType | **Attribute description:** |
| Represents current action type, chosen by the user to be operated on selected file or folder. |
| **Program description language:** |
| private ACTION\_TYPE m\_currActionType; |
| QModelIndex m\_processedIndex | **Attribute description:** |
| This is a declaration for QModelIndex, that is used to locate data in a selected data model. |
| **Program description language:** |
| private QModelIndex m\_processedIndex; |
| ACTIVE\_SIDE m\_processedSide | **Attribute description:** |
| This is a declaration of an ACTIVE\_SIDE object that represents the window, whose data model will be changed. |
| **Program description language:** |
| private ACTIVE\_SIDE m\_processedSide; |
| **Methods(operations):** | **Methods descriptions:** |
| activateModel() | A method to set m\_activeSide according to current active side, chosen by user, retrieved from m\_uiHandler. |
| **Program Description language:** |
| void FileModelHandler::**activateModel**(ACTIVE\_SIDE activeSide)  {  m\_activeSide = activeSide;  } |
| makeDir() | **Methods descriptions:** |
| A method to create a new folder in selected parent folder. |
| **Program Description language:** |
| void FileModelHandler::**makeDir**(QModelIndex index, QString name)  {  *if*(!index.isValid())  {  *return*;  }  *if*(m\_activeSide == ACTIVE\_SIDE::*LEFT*)  {  m\_modelLeft->mkdir(index, name);  }  … |
| deleteItem() | **Methods descriptions:** |
| A method to delete selected file or directory. |
| **Program Description language:** |
| void FileModelHandler::**deleteItem**(QModelIndex index)  {  *if*(!index.isValid())  {  *return*;  }  *if*(m\_activeSide == ACTIVE\_SIDE::*LEFT*)  {  m\_modelLeft->remove(index);  }  … |
| initCopy() | **Methods descriptions:** |
| Memorizes the type of operation (copy), index of proceeded file and current side, where is the chosen file. |
| **Program Description language:** |
| void FileModelHandler::**initCopy**(QModelIndex index)  {  *if*(!index.isValid())  {  *return*;  }  m\_processedSide = m\_activeSide;  m\_processedIndex = index;  m\_currActionType = ACTION\_TYPE::*COPY*;  } |
| initMove() | **Methods descriptions:** |
| Memorizes the type of operation (move), index of proceeded file and current side, where is the chosen file. |
| **Program Description language:** |
| void FileModelHandler::**initMove**(QModelIndex index)  {  *if*(!index.isValid())  {  *return*;  }  m\_processedSide = m\_activeSide;  m\_processedIndex = index;  m\_currActionType = ACTION\_TYPE::*MOVE*;  } |
| insert() | **Methods descriptions:** |
| Inserts selected file or directory to the selected directory, after copying or moving those items. |
| **Program Description language:** |
| void FileModelHandler::**insert**(QModelIndex index)  {  *if*(m\_currActionType == ACTION\_TYPE::*NEITHER*)  {  *return*;  }  *if*(!index.isValid())  {  *return*;  }  *if*(m\_currActionType == ACTION\_TYPE::*COPY*)  {  QString str = "Copying in progress";  QString filePathFrom, filePathTo;  QFileSystemModel \* processedModel, \* currModel;  *if*(m\_processedSide == ACTIVE\_SIDE::*LEFT*)  {  processedModel = m\_modelLeft;  }  … |
| copyPath() | **Methods descriptions:** |
| A method to copy path from source to destination, used to copy directories recursively. |
| **Program Description language:** |
| void FileModelHandler::**copyPath**(QString src, QString dst)  {  QDir dir(src);  QFileInfo srcInfo(src);  *if* (! dir.exists())  {  *return*;  }  dir.mkpath(dst + QDir::separator() + srcInfo.fileName());  foreach (QString d, dir.entryList(QDir::Dirs | QDir::NoDotAndDotDot))  {  QString dst\_path = dst + QDir::separator() + srcInfo.fileName();  … |
| moveDragAndDrop() | **Methods descriptions:** |
| A method that handles drag and drop with progress bar, does moving files and directories. |
| **Program Description language:** |
| void FileModelHandler::**moveDragAndDrop**(QModelIndex indexFrom, QModelIndex indexTo, ACTIVE\_SIDE indexToSide)  {  QString str = "Moving in progress";  QString filePathFrom, filePathTo;  QFileSystemModel \* processedModel, \* currModel;  *//processedModel* *==* *destination* *model*  *if*(indexToSide == ACTIVE\_SIDE::*LEFT*)  {  processedModel = m\_modelLeft;  }  *Else*  … |

**2.1.2 UIHandler**

|  |  |
| --- | --- |
| **Class name:** UIHandler | |
| **Brief description:** this class is responsible for controlling two QTreeViews (right and left) that are used to represent data provided by QFileSystemModule in a tree form. | |
| **Attributes(fields):** | **Attribute description:** |
| MainWindow \*ui | MainWindow defines the UI :: MainWindow class  in the namespace of the UI. This class  generated automatically based on what was created in forms. It is a pointer to the interface builder object. |
| **Program description language:** |
| private Ui::MainWindow \*ui; |
| QMap<int, ProgressBarHandler\*> progressBarMap | **Attribute description:** |
| Map that represents progress bars started at the same time. |
| **Program description language:** |
| private QMap<int, ProgressBarHandler\*> progressBarMap; |
| **Methods(operations):** | **Methods descriptions:** |
| getActiveSide() | A method that retrieves active side using the information about the latest click of user on one of the trees. |
| **Program Description language:** |
| ACTIVE\_SIDE UiHandler::**getActiveSide**(QObject \*treeObj)  {  *if*(treeObj == ui->treeViewLeft)  {  qDebug() << "Left tree is active";  *return* ACTIVE\_SIDE::*LEFT*;  }  *else* *if*(treeObj == ui->treeViewRight)  {  qDebug() << "Right tree is active";  *return* ACTIVE\_SIDE::*RIGHT*;  }  *return* ACTIVE\_SIDE::*NEITHER*;  } |
| getModelIndex() | **Methods descriptions:** |
| A method that returns the index of the object that was clicked on one of the tree views. |
| **Program Description language:** |
| QModelIndex UiHandler::**getModelIndex**(ACTIVE\_SIDE activeSide)  {  *if*(activeSide == ACTIVE\_SIDE::*LEFT*)  {  *return* ui->treeViewLeft->currentIndex();  }  *else* *if*(activeSide == ACTIVE\_SIDE::*RIGHT*)  {  *return* ui->treeViewRight->currentIndex();  }  *else*  {  *return* QModelIndex();  }  } |
| isDeletingConfirmed() | **Methods descriptions:** |
| A method that gets confirmation of deletion item from user using QMessageBox. |
| **Program Description language:** |
| bool UiHandler::**isDeletingConfirmed**()  {  QMessageBox::StandardButton reply;  reply = QMessageBox::information(ui->centralwidget, "Delete",  "Do you want to delete selected item?",  QMessageBox::*Yes* | QMessageBox::*No*);  *if*(reply == QMessageBox::*Yes*)  {  *return* *true*;  }  *else*  {  *return* *false*;  }  } |
| setAttributeShown() | **Methods descriptions:** |
| A method that hides or shows column, clicked by user. |
| **Program Description language:** |
| void UiHandler::**setAttributeShown**(ATTRIBUTE\_NAME name)  {  *switch*(name)  {  *case* ATTRIBUTE\_NAME::*DATE*:  ui->treeViewLeft->setColumnHidden(3, !ui->actionShowDate->isChecked());  ui->treeViewRight->setColumnHidden(3, !ui->actionShowDate->isChecked());  *break*;  … |
| progressBarStart() | **Methods descriptions:** |
| Public slot used to create a new progress bar. |
| **Program Description language:** |
| void UiHandler::**progressBarStart**(QString operationame, int operationIndex)  {  ProgressBarHandler \*timer = *new* ProgressBarHandler(operationame);  timer->show();  progressBarMap.insert(operationIndex, timer);  } |
| progressBarEnd() | **Methods descriptions:** |
| Public slot used to end progress bar. |
| **Program Description language:** |
| void UiHandler::**progressBarEnd**(int operationIndex)  {  progressBarMap[operationIndex]->close();  *delete* progressBarMap[operationIndex];  progressBarMap.remove(operationIndex);  } |
| progressBarUpdate() | **Methods descriptions:** |
| Public slot used to update the value of progress bar. |
| **Program Description language:** |
| void UiHandler::**progressBarUpdate**(int operationIndex, int progressValue)  {  progressBarMap[operationIndex]->updateProgress(progressValue);  } |
| setModels() | **Methods descriptions:** |
| A method that sets specific models to the corresponding tree views. |
| **Program Description language:** |
| void UiHandler::**setModels**(FileModelHandler \*modelHandler)  {  ui->treeViewLeft->*setModel*(modelHandler->getLeftModel());  ui->treeViewRight->*setModel*(modelHandler->getRightModel());  ui->treeViewLeft->header()->setSectionResizeMode(0, QHeaderView::*ResizeToContents*);  ui->treeViewRight->header()->setSectionResizeMode(0, QHeaderView::*ResizeToContents*);  } |

**2.1.3 MainWindow**

|  |  |
| --- | --- |
| **Class name:** MainWindow | |
| **Brief description:** this class is responsible for controlling the interaction between the objects of UIHandler and FileModelHandler classes, and it reacts to user actions. | |
| **Attributes(fields):** | **Attribute description:** |
| UiHandler \*m\_uiHandler | An object of UIHandler class. |
| **Program description language:** |
| private UiHandler \*m\_uiHandler; |
| FileModelHandler \*m\_modelHandler | **Attribute description:** |
| An object of FileModelHandler class. |
| **Program description language:** |
| private FileModelHandler \*m\_modelHandler; |
| **Methods(operations):** | **Methods descriptions:** |
| on\_exitButton\_clicked() | Public slot that responds when user clicks on exit button. |
| **Program Description language:** |
| void MainWindow::**on\_exitButton\_clicked**()  {  *this*->close();  } |
| on\_folderButton\_clicked() | **Methods descriptions:** |
| Public slot that responds when user clicks on New folder button. |
| **Program Description language:** |
| void MainWindow::**on\_folderButton\_clicked**()  {  QModelIndex index = m\_uiHandler->getModelIndex(m\_modelHandler->getActiveSide());  QString name = QInputDialog::getText(*this*, "New folder name", "Enter the folder name:");  *if*(name.isEmpty()) *return*;  m\_modelHandler->makeDir(index, name);  } |
| eventFilter() | **Methods descriptions:** |
| A method that activates corresponding model when specific treeView is activated by user mouse click. |
| **Program Description language:** |
| bool MainWindow::***eventFilter***(QObject \*obj, QEvent \*event)  {  *if* (event->type() == QEvent::*FocusIn*)  {  m\_modelHandler->activateModel(m\_uiHandler->getActiveSide(obj));  }  *//* *standard* *event* *processing*  *return* QObject::eventFilter(obj, event);  } |

**2.1.4 ProgressBarHandler**

|  |  |
| --- | --- |
| **Class name:** ProgressBarHandler | |
| **Brief description:** this class is responsible for handling progress bar. | |
| **Attributes(fields):** | **Attribute description:** |
| QProgressBar \*progressBar | A widget that provides a horizontal progress bar. |
| **Program description language:** |
| private QProgressBar \*progressBar; |
| **Methods(operations):** | **Methods descriptions:** |
| updateProgress() | A method that changes the value of progress bar according to the value, provided be signal that was emitted by the object of FileModelHandler class. |
| **Program Description language:** |
| void ProgressBarHandler::**updateProgress**(int updateValue)  {  progressBar->setValue(progressBar->value()+updateValue);  } |

**2.1.5 QTreeViewWithProgressBar**

|  |  |
| --- | --- |
| **Class name:** QTreeViewWithProgressBar | |
| **Brief description:** this class is responsible for handling progress bar when user drags and drops objects. | |
| **Attributes(fields):** | **Attribute description:** |
| FileModelHandler\* m\_modelHandler | An object of FileModelHandler class that provides data model for the local filesystem. |
| **Program description language:** |
| private FileModelHandler\* m\_modelHandler; |
| ACTIVE\_SIDE m\_side; | **Attribute description:** |
| Represents current active side, the window (left or right) which was clicked by user for the last time. |
| **Program description language:** |
| private ACTIVE\_SIDE m\_side; |
| **Methods:** | **Methods descriptions:** |
| setModelHandler() | A method that sets m\_modelHandler to the model, which is currently proceeded. |
| **Program Description language:** |
| void QTreeViewWithProgressBar::**setModelHandler**(FileModelHandler \*model)  {  m\_modelHandler = model;  } |
| setSide () | **Methods descriptions:** |
| A method that sets m\_side to active side, selected by user. |
| **Program Description language:** |
| void QTreeViewWithProgressBar::**setSide**(ACTIVE\_SIDE side)  {  m\_side = side;  } |
| dropEvent() | **Methods descriptions:** |
| This event handler is called when the drag is dropped on this widget. The event is passed in the event parameter. |
| **Program Description language:** |
| void QTreeViewWithProgressBar::***dropEvent***(QDropEvent \*e)  {  QModelIndex index = *this*->*indexAt*(e->pos());  *if*(!index.isValid())  {  *return*;  }  foreach (*const* QUrl &url, e->mimeData()->urls()) {  QString fileName = url.toLocalFile();  qDebug() << "Dropped file:" << fileName;  m\_modelHandler->moveDragAndDrop(  m\_modelHandler->getIndexByFilePath(fileName),  index,  m\_side);  }  } |

**3. Enum class description.**

ACTIVE\_SIDE enum class represents existing options of active windows, clicked by user.

*enum* *class* **ACTIVE\_SIDE**

{

*LEFT*,

*RIGHT*,

*NEITHER*

};

ACTION\_TYPE enum class represents existing options operations, that are possible to apply on selected file or folder.

*enum* *class* **ACTION\_TYPE**

{

*COPY*,

*MOVE*,

*NEITHER*

};

ATTRIBURE\_NAME enum class represents existing attributes of local file system, that can be shown or hided.

*enum* *class* **ATTRIBUTE\_NAME**

{

*SIZE*,

*DATE*,

*TYPE*

};