JFace @ TUES

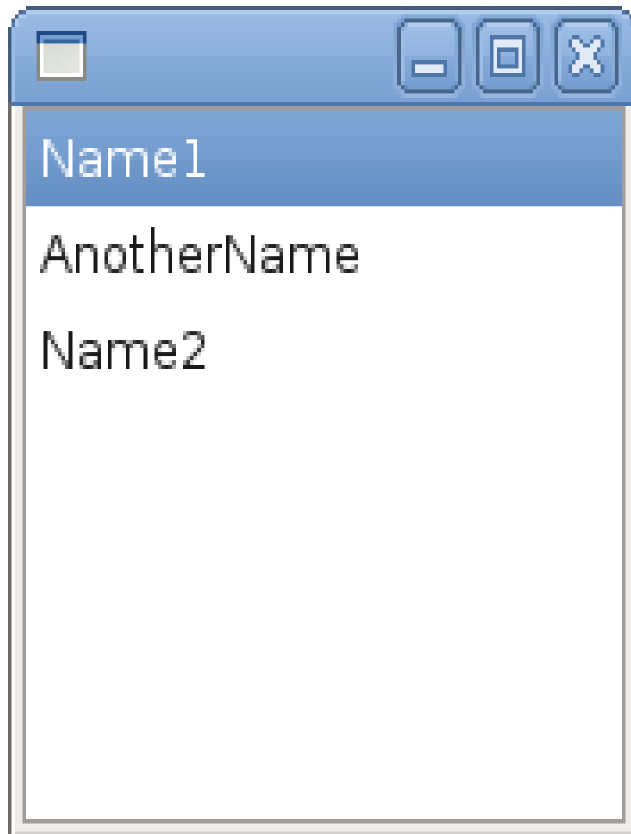
Кирил Митов @ TUES

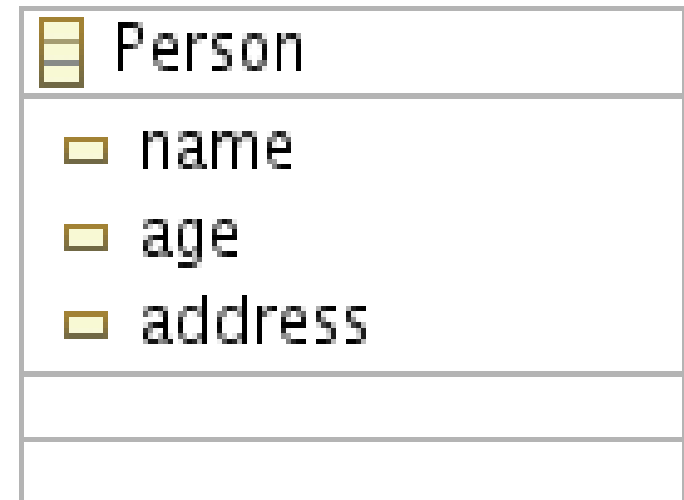
Същност

- **SWT предоставя много възможности за развитие на потребителския интерфейс.**
- **Неудобството се изразява в това, че при представяне на данните се използват прости типове като *низове, числа, картинки*. Това е неудобно при работа с обектно ориентиран език като Java.**

Проблем

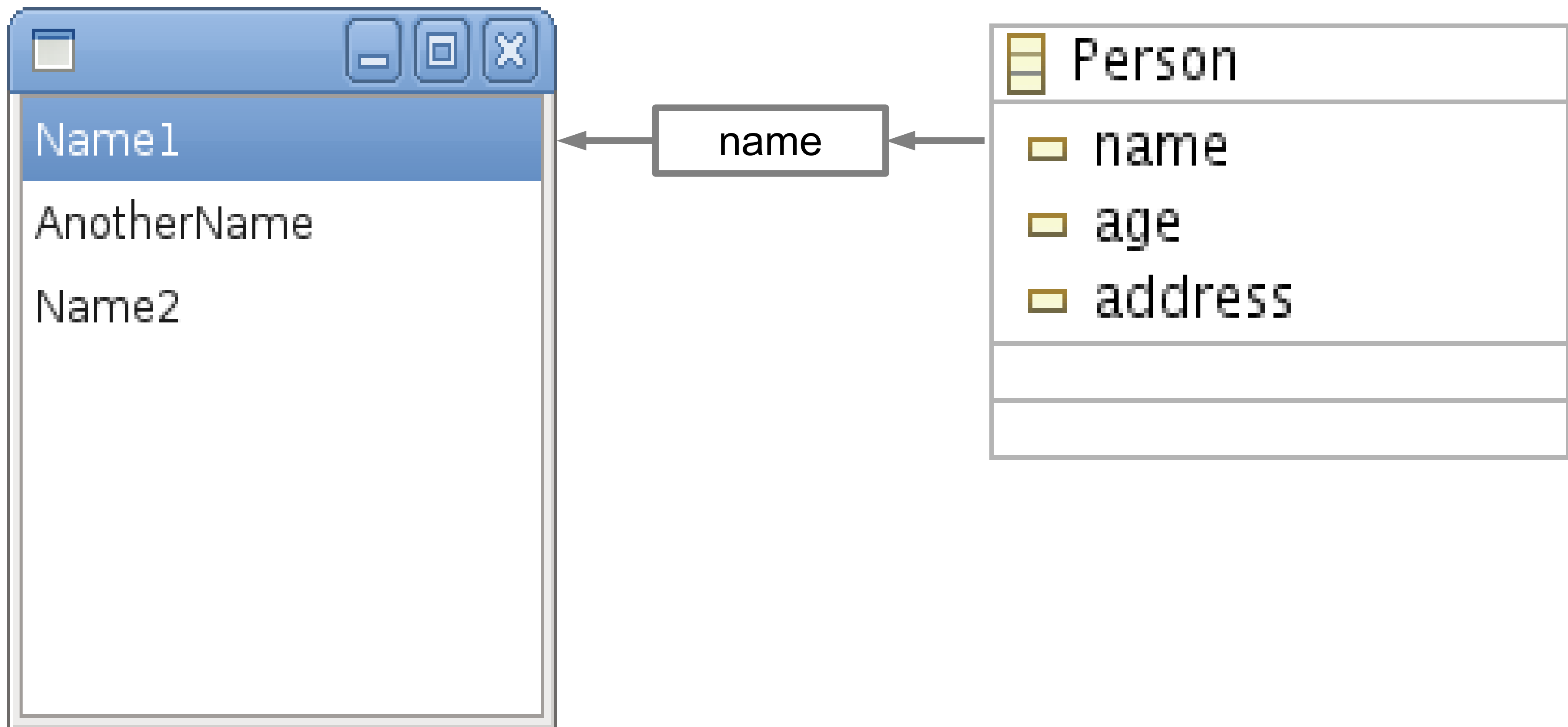
Трябва да покажете полето `name` на всеки обект `Person`





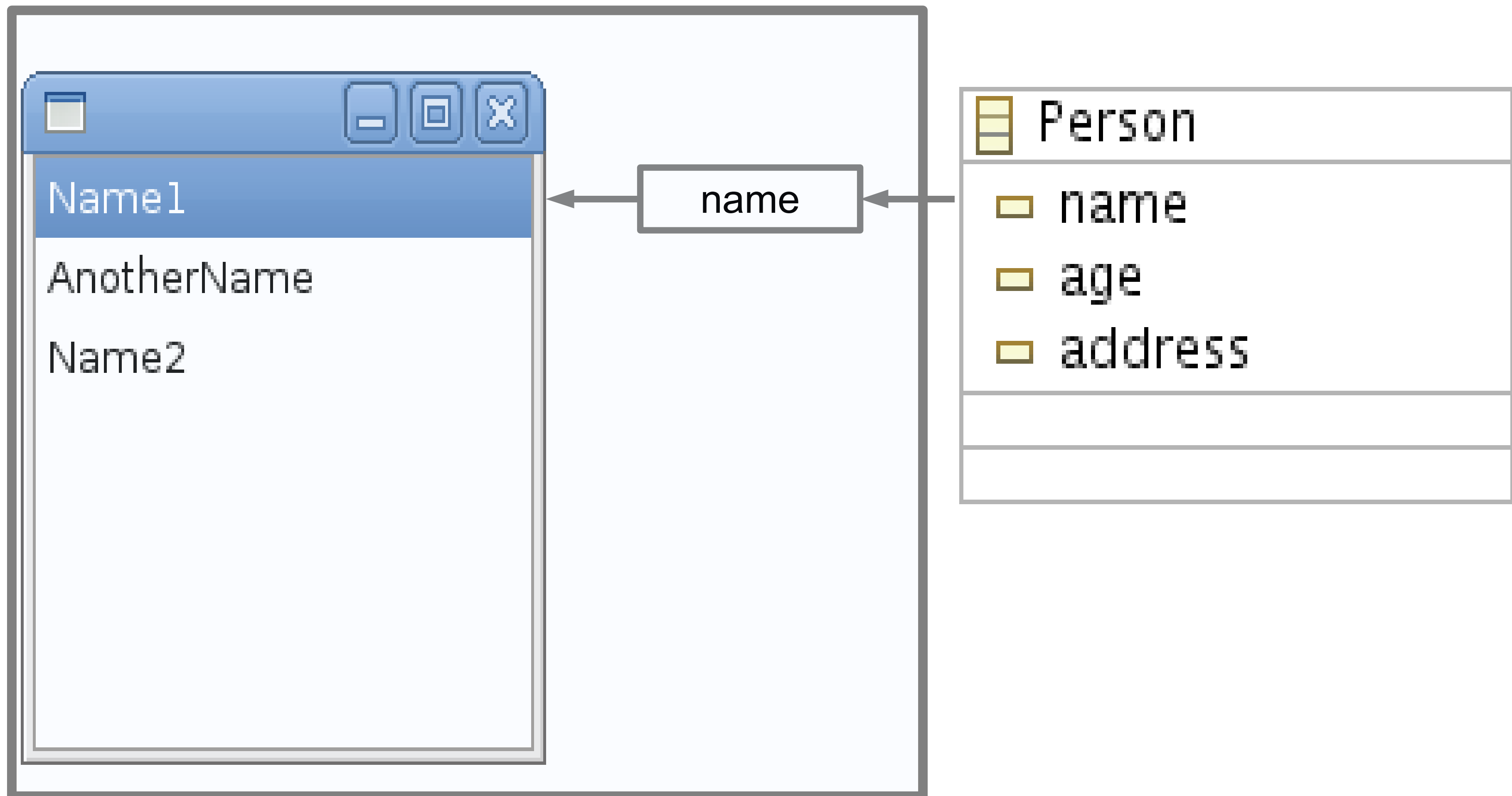
Проблем

Извличате стойността на полето `name` и я поставяте в списъка



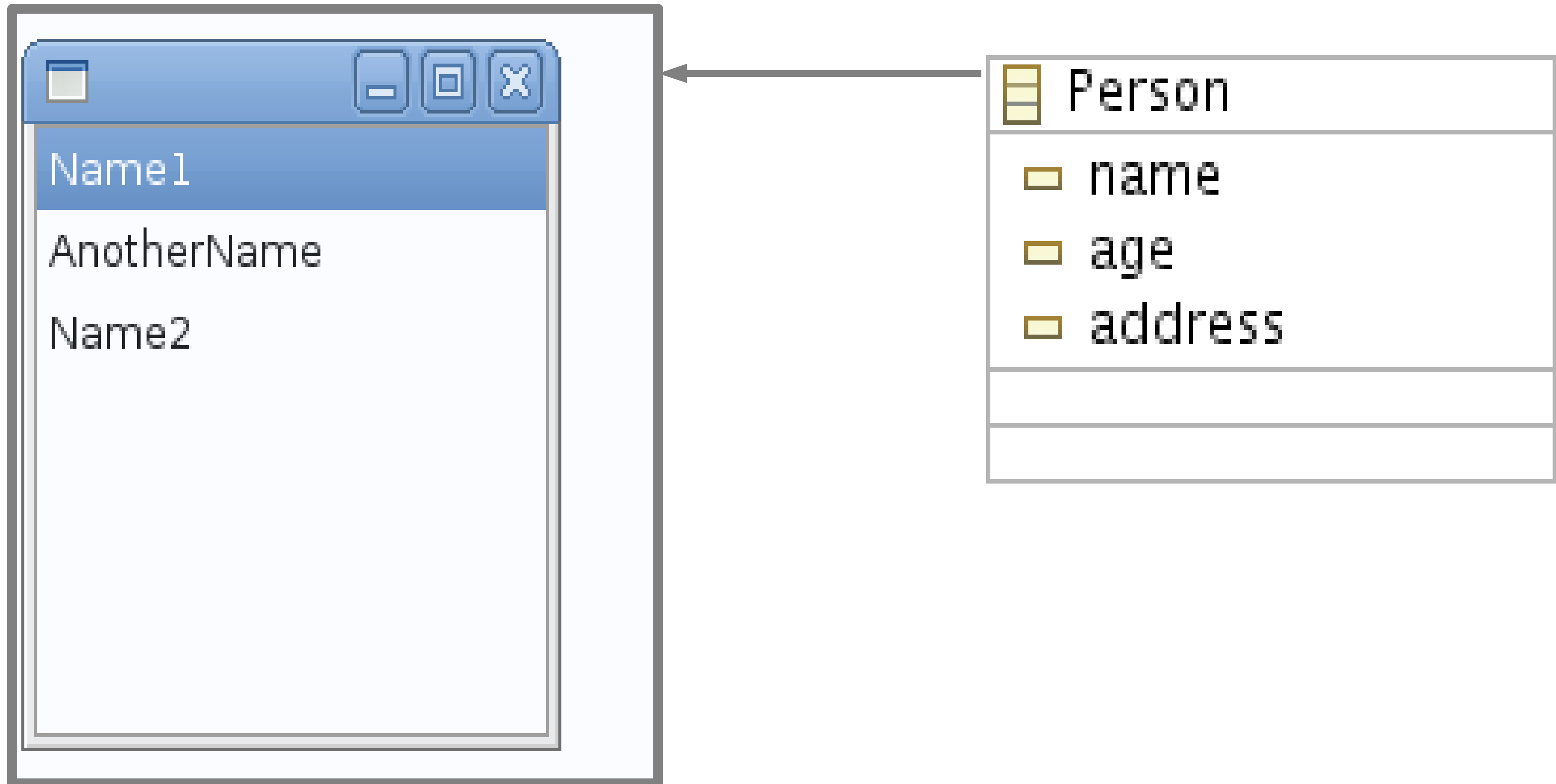
Проблем

Не може ли някой друг да свърши тази работа!



Решение

Проблемът може да се реши чрез иползването на JFace

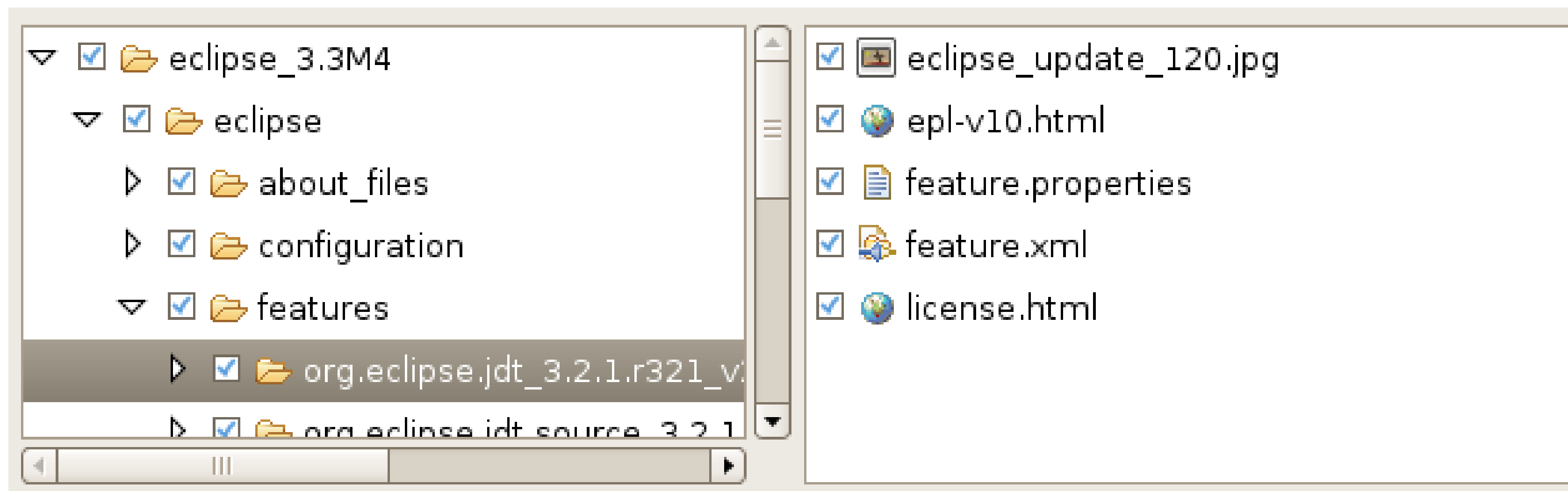


Същност

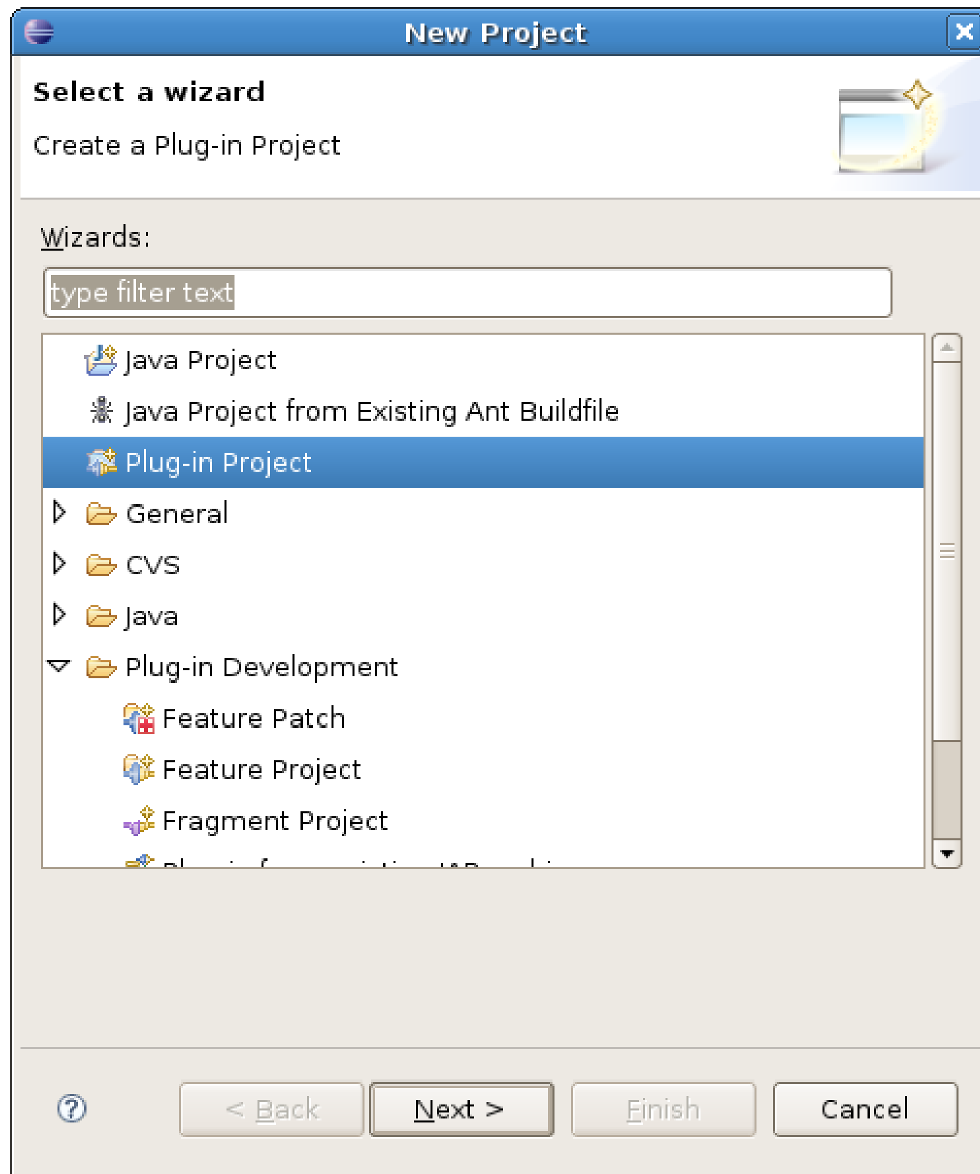
- Представянето на обектите се улеснява значително с използването на JFace
- JFace представлява библиотека улесняваща извършването на често срещани задачи при разработката на потребителски интерфейс.
- JFace работи със SWT без да се опитва да го замести.

Същност

- Интересна характеристика на JFace е наличието на така наречените „viewers“.
- Viewer-ите са обекти адаптиращи обектно-ориентирания модел към потребителския интерфейс. Целта е да се улесни изграждането на често срещани графични компоненти като структурирани списъци, таблици, дървета.
- Пример за viewer може да бъде видян на фигурата.



Същност

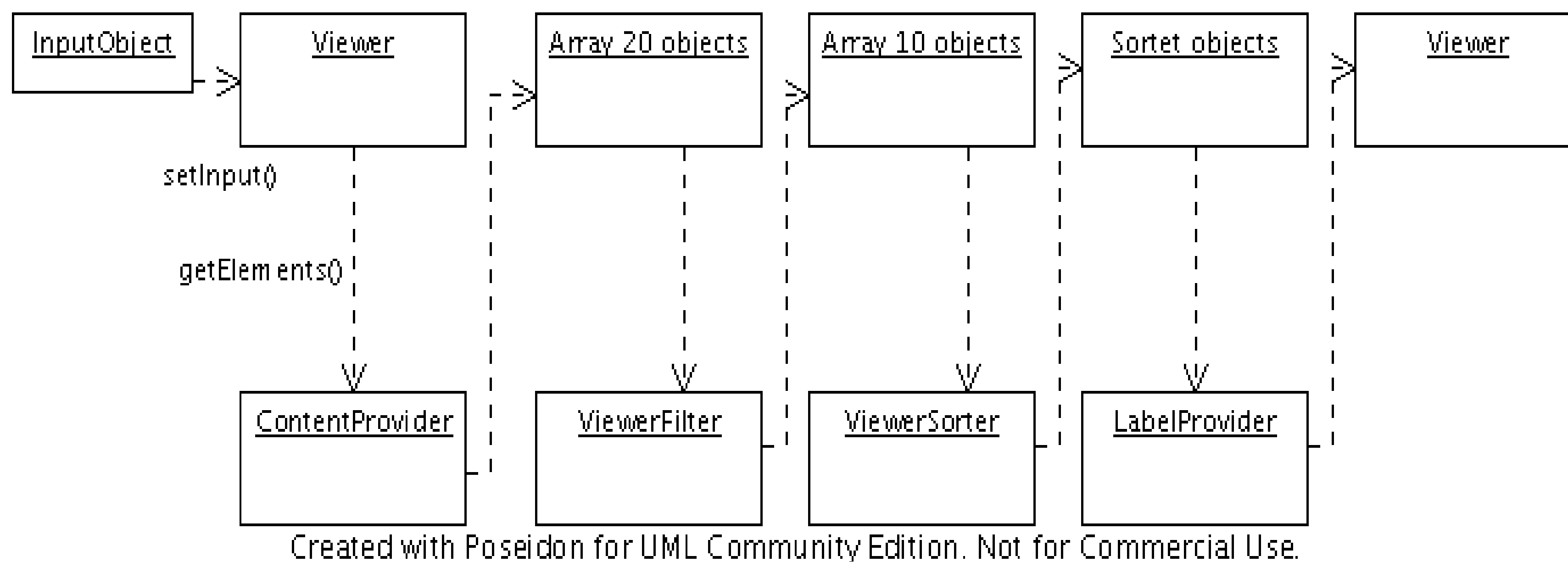


- **Всичко, което вижда потребителят представлява SWT компоненти.**
- **JFace помага за по-лесната организация на информацията**

Същност

- JFace viewer-ите позволяват директната работа със сложни обекти без да се налага на програмиста да представя тези обекти като съвкупност от низови и числа. Това се извършва автоматично.

- За да се отдели потребителският интерфейс от бизнес логиката на приложението се въвеждат няколко допълнителни класа.

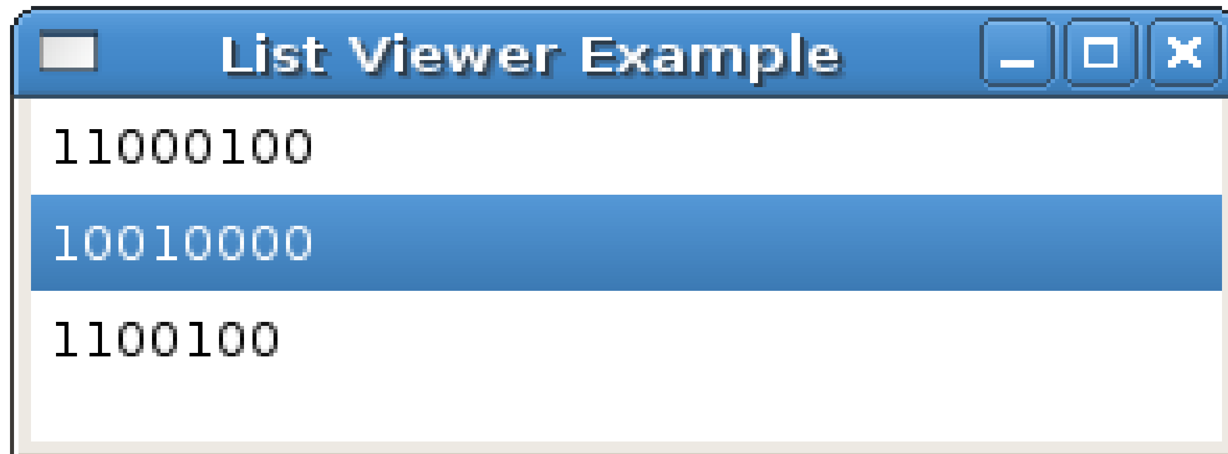


Същност

- **За да съществува обектът viewer трябва да са зададени съответните допълнителни класове**
  - **Задължителни – ContentProvider, LabelProvider**
  - **Незадължителни - ViewerFilter, ViewerComparator**

- **Пример:**
  - **Даден е масив от цели числа. (inputObject)**
  - **Искаме да изобразим елементите на масива във viewer структуриран като списък. За целта ще използваме ListViewer. (ContentProvider)**
  - **Трябва да се покажат само четните числа (ViewerFilter)**
  - **Числата трябва да са сортирани в низходящ ред (ViewerComparator)**
  - **Числата трябва да са показани в двоичен код (LabelProvider)**

Пример



ContentProvider

Всеки Viewer се нуждае от съдържание, което да покаже. Това съдържание се извлича от модела. Извличането става с помоща на специален обект от тип
`org.eclipse.jface.viewers.IContentProvider`
или неговите най-често използвани наследници
`org.eclipse.jface.viewers.IStucturedContentProvider`
`org.eclipse.jface.viewers.ITreeContentProvider`

```
public interface IStructuredContentProvider extends
IContentProvider {
    public Object[] getElements(Object inputElement);
}
```

Методът getElements() връща масив от обекти, извлечени от входния елемен inputElement.

ContentProvider

```
public interface ITreeContentProvider extends
IStructuredContentProvider {
   public Object[] getChildren(Object parentElement);
   public Object getParent(Object element);
   public boolean hasChildren(Object element);
}
```

- getChildren() - резултатът от метода е масив от елементи. Тези елементи ще бъдат изобразени като деца на parentElement.
- getParent() - в повечето случаи смислената му имплементация не е необходима. Целта е като резултат да се върне родителят на посочения елемент.
- hasChildren() - при работа с ITtreeContentProvider първо се вика hasChildren() и ако резултатът е true се вика getChildren()

ContentProvider

Съгласно примера входният елемен е масив с цели числа. Числата ще бъдат представени като обекти от тип Integer.

```
public class NumbersContentProvider implements
IStructuredContentProvider {
   public Object[] getElements(Object inputElement) {
      if (inputElement instanceof int[]) {
         int[] array = (int[]) inputElement;
         Integer[] result = new Integer[array.length];
         for (int i = 0; i < array.length; i++) {
            result[i] = new Integer(array[i]);
         }
         return result;
      }
      return null;
   }};
```

Задаването на ContentProvider става по следния начин.

```
final StructuredViewer viewer = new TreeViewer(shell,
SWT.SINGLE);
viewer.setContentProvider(new NumbersContentProvider());
```

ViewerFilter

Съдържанието на viewer-а е определено с помощта на ContentProvider-а. Следва филтриране на съдържанието.
Всеки viewer може да има един или няколко филтъра. Филтърът представлява обект от тип `org.eclipse.jface.viewers.ViewerFilter`.
Добавянето на филтър към viewer става с помощта на метода `StructuredViewer.addFilter()`.

```
public abstract class ViewerFilter {
  /* code missed  */

    public abstract boolean select(Viewer viewer, Object 
parentElement,
            Object element);
}
```

ViewerFilter

Основният метод е select(). Методът връща true ако даденият елемент минава през филтъра – т.е не се филтрира.
Логиката на филтрация е отново напълно произволна и зависи само от програмиста.
Съгласно разработвания пример трябва да се показват само елементите, които са четни.

```
public class NumbersFilter extends ViewerFilter {
   @Override
   public boolean select(Viewer viewer, Object
parentElement, Object element) {
      if (element instanceof Integer) {
         int value = ((Integer) element).intValue();
         if (value % 2 == 0)
            return true;
      }
      return false;
   }
}
```

ViewerFilter

Добавянето на филтър става по следния начин:

```
final StructuredViewer viewer = new ListViewer(shell,
SWT.SINGLE);
viewer.addFilter(new NumbersFilter());
```

```
final StructuredViewer viewer = new ListViewer(shell,
SWT.SINGLE);
viewer.addFilter(new ViewerFilter(){
   @Override
   public boolean select(Viewer viewer, Object
parentElement, Object element) {
      if (element instanceof Integer) {
         int value = ((Integer) element).intValue();
         if (value % 2 == 0)
            return true;
      }
      return false;
   }
});
```

ViewerComparator

С помощта на ContentProvider-а е определено съдържанието. Следва стъпка на сортиране. Ако използвания viewer го позволява е възможно задаването на определен алгоритъм на сортиране на елементите. Сортирането се извършва с помощта на обект от тип `org.eclipse.jface.viewers.ViewerComparator.`

```
public class ViewerComparator {
   /* .  .  .  */

    public int category(Object element) {
        return 0;
    }

    public int compare(Viewer viewer, Object e1, Object
e2) {
   /* .  .  .  */
   }
}
```

ViewerComparator

По подразбиране алгоритъмът на сортиране се извършва на две стъпки:

- **Първо елементите се групират по категории и ранк от 0 до n чрез**

```
public int category(Object element) {
        return 0;
}
```

- **След това се извършва сортиране на елементите в категорията посредством текстовата презентация на обектите – т.е сортират се в азбучен ред.**

ViewerComparator

Съгласно използвания пример числата трябва да са сортирани в низходящ ред.

```
public class NumbersComparator extends ViewerComparator {
   @Override
   public int compare(Viewer viewer, Object e1, Object e2)
{
      if ((e1 instanceof Integer)
           && (e2 instanceof Integer)) {
         return ((Integer) e2).compareTo((Integer) e1);
      }
      throw new IllegalArgumentException("The comparator
compares only Integers");
   }
}
```

Задаване на обект от тип ViewerComparator

```
final StructuredViewer viewer = new ListViewer(shell,
SWT.SINGLE);
viewer.setComparator(new NumbersComparator());
```

LabelProvider

Съдържанието на viewer-а е определено. Елементите са филтрирани и сортирани, но все още не са показани на потребителя.
JFace viewer-ите използват SWT компоненти за представяне на информацията. Следователно трябва да се извърши преобразуване на елементите от съдържанието към елементи от тип String, Image и т.н.
Това преобразуване се извършва с помощта на обект от тип `org.eclipse.jface.viewers.ILabelProvider.`

```
public interface ILabelProvider extends IBaseLabelProvider
{

   public Image getImage(Object element);

   public String getText(Object element);
}
```

Интерфейсът има два основни метода – getImage() и getText(). Резултатът върнат от тези методи ще бъде показан на потребителя.

LabelProvider

Важно!!!. Не всички входни елементи трябва да се поддържат. Необходимо е еднотипно решение.

```
public String getText(Object element) {
   if (element instanceof java.util.ArrayList) {
      return "java.util.ArrayList";
   } else if (element instanceof java.util.LinkedList) {
      return "java.util.LinkedList";
   }
   thrown new IllegalArgumentException();
   /* A solution is to throw an
    * IllegalArgumentException()
    *
    */
}
```

LabelProvider

Съгласно примера трябва да се показват двоичните кодове на съответните числа.

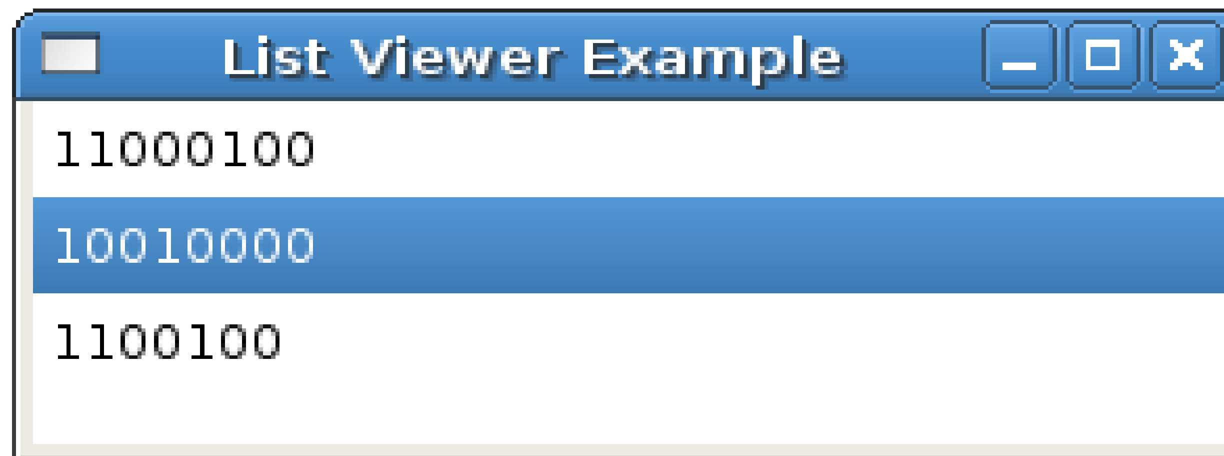
```
public class NumbersLabelProvider extends LabelProvider {
    @Override
    public String getText(Object element) {
        if (element instanceof Integer) {
            int value = ((Integer) element).intValue();
            return Integer.toBinaryString(value);
        }
        throw new IllegalArgumentException("This label
provider supports only Integers");
    }
}
```

Пример за задаване на LabelProvider.

```
final StructuredViewer viewer = new TreeViewer(shell,
SWT.SINGLE);
viewer.setLabelProvider(new NumberLabelProvider());
```

ListViewer

```
int[] input = new int[] { 100, 121, 144, 169, 196 };
final StructuredViewer viewer = new ListViewer(shell,
SWT.SINGLE);
viewer.setContentProvider(new NumbersContentProvider());
viewer.setLabelProvider(new NumbersLabelProvider());
viewer.addFilter(new NumbersFilter());
viewer.setComparator(new NumbersComparator());
viewer.setInput(input);
```

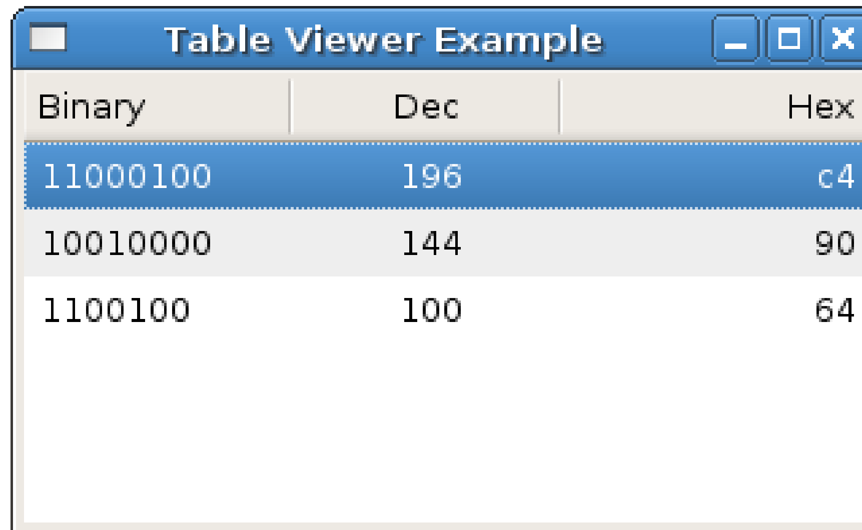


TableViewer

- Предоставя изглед на таблица за съответното съдържание.

- Необходими са contentProvider, labelProvider и определен набор от колони, в които да се показва информацията.

Пример:
Представяне на число в двоичен, десетичен и шестнадесетичен вид. Целта е максимално лесно да адаптираме предходния пример към новите изисквания.



TableViewer

За да работи един TableViewer правилно е необходимо да са посочени неговите колони и техните имена. Инициализацията се извършва по следния начин.

```
1  final TableViewer viewer = new TableViewer(shell,
SWT.SINGLE);
2  viewer.getTable().setHeaderVisible(true);
3  viewer.getTable().setLinesVisible(true);
4  String[] columnNames = new String[] { "Binary", "Dec",
                                          "Hex" };
5  int[] columnAlignments = new int[] { SWT.LEFT,
                              SWT.CENTER, SWT.RIGHT };
6  for (int i = 0; i < columnNames.length; i++) {
7     TableColumn tableColumn = new
8        TableColumn(viewer.getTable(),
                    columnAlignments[i]);
9     tableColumn.setText(columnNames[i]);
10    tableColumn.setWidth(100);
11 }
```

TableViewer

```
2   viewer.getTable().setHeaderVisible(true);
3   viewer.getTable().setLinesVisible(true);
4   String[] columnNames = new String[] { "Binary", "Dec",
                                          "Hex" };
5   int[] columnAlignments = new int[] { SWT.LEFT,
                           SWT.CENTER, SWT.RIGHT };
```

Ред 2: Имената на колоните ще са видими.

Ред 3: Редовете на таблицата ще са оцветени във сиво през ред.

Ред 4: Създаване на масив използван за задаване имената на колоните.

Ред 5: Масив определящ подравняването във всяка колона.

TableViewer

```
6  for (int i = 0; i < columnNames.length; i++) {
7      TableColumn tableColumn = new
          TableColumn(viewer.getTable(),
                      columnAlignments[i]);
8      tableColumn.setText(columnNames[i]);
9      tableColumn.setWidth(100);
10 }
```

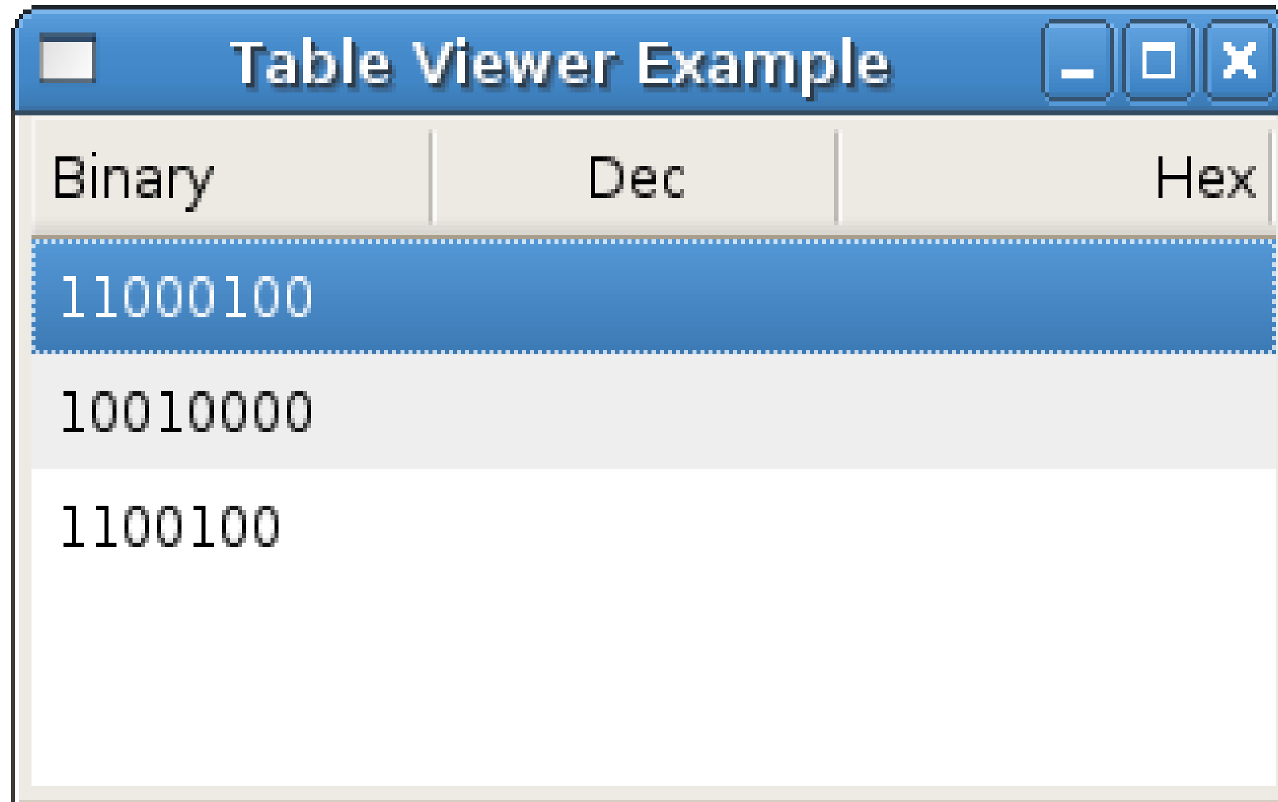
Ред 7: Създаване на нов обект от тип TableColumn. Колоните в таблицата представляват деца на таблицата.

Ред 8: Задаване името на колоната.

Ред 9: Задаване широчината на колоната.

TableViewer

В този момент приложението изглежда следния начин:



TableViewer

Съгласно предходния пример – класовете, които няма да променяме са

- **NumbersContentProvider – съдържанието, което искаме да показваме е същото – масив от числа**
- **NumbersFilter – искаме на филтрираме числата по същият признак – т.е по четност**
- **NumbersComparator – числата отново трябва да са сортирани по низходящ ред.**

Тъй като viewer-ът в момента разполага с няколко колони е необходимо да се зададе текст за всяка една от клетките. Следователно промяната трябва да се извърши в NumbersLabelProvider класа.
NumbersLabelProvider трябва да имплементира интерфейса `org.eclipse.jface.viewers.ITableLabelProvider`

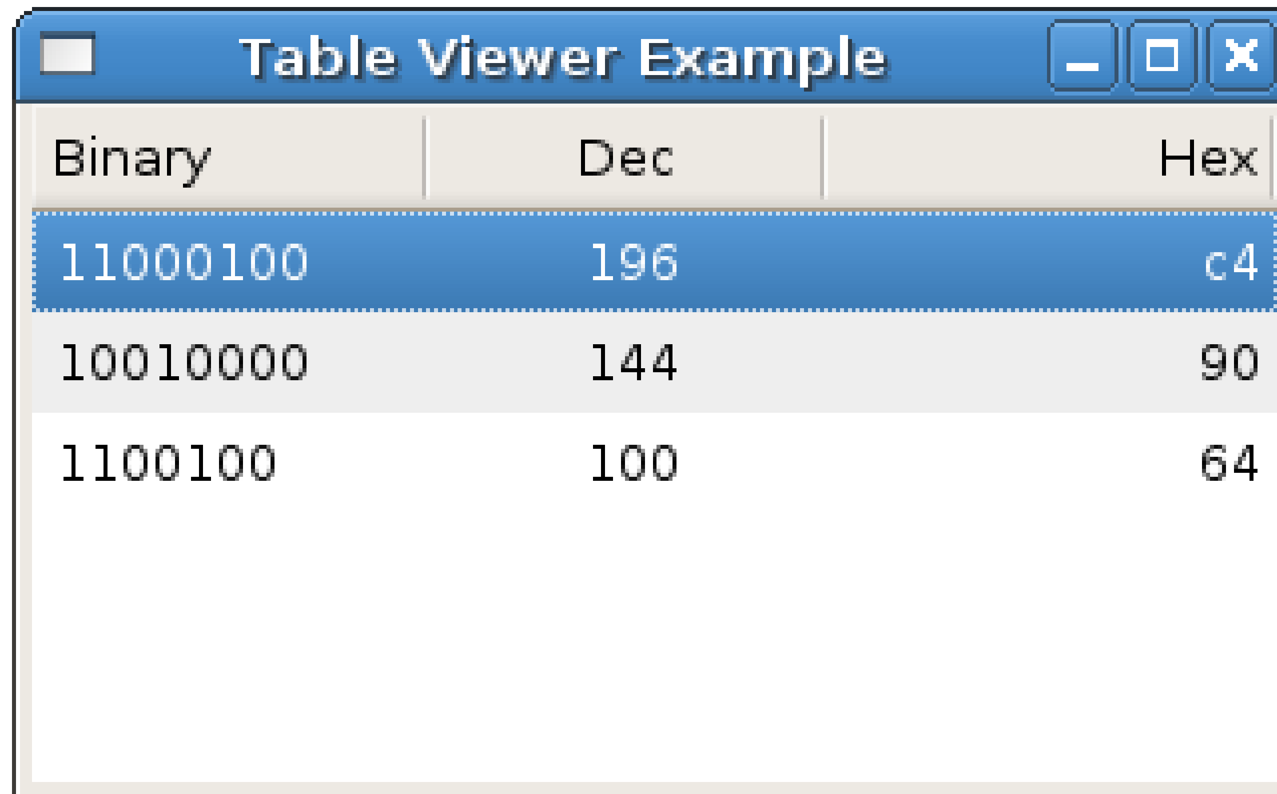
TableViewer

```
public class NumbersLabelProvider extends LabelProvider
implements ITableLabelProvider {
   public String getColumnText(Object element,
                                         int columnIndex) {
      if (!(element instanceof Integer)) {
         throw new IllegalArgumentException("This is a
                     label provider only for Integers");
      }
      int value = ((Integer) element).intValue();
      switch (columnIndex) {
      case 0:return Integer.toBinaryString(value);
      case 1:return Integer.toString(value);
      case 2:return Integer.toHexString(value);
      }
      return null;
   }
}
```

Методът getColumnText() връща резултат спрямо колоната определена с columnIndex

TableViewer

Резултатът от реализацията на примера е показа на фигурата.



TreeViewer

- Предоставя възможност за изобразяване на дървовидна структура.

- Необходими са contentProvider, labelProvider, входни данни.

Целта на примера е да модифицираме ListViewer-a, така че като деца на всяко число да се показват резултатът от разделянето му на десет.
Класовете които няма да се променят са:

- NumbersLabelProvider
- NumbersComparator
- NumbersFilter

Промяната се извършва единствено в NumbersContentProvider.
Тъй като предоставяното съдържание трябва да е в дървовидна структура то NumbersContentProvider трябва да имплементира `org.eclipse.jface.viewers.ITreeContentProvider`

TreeViewer

```
public class NumbersContentProvider implements
ITreeContentProvider {

      public Object[] getElements(Object inputElement) {
         return getChildren(inputElement);
      }
      public void dispose() {}
      public void inputChanged(Viewer viewer, Object
            oldInput, Object newInput) {}
      public Object[] getChildren(Object parentElement) {
         /*explained later*/
      }
      public Object getParent(Object element) {
         return null;
      }
      public boolean hasChildren(Object element) {
         return getChildren(element).length > 0;
      }
}
```

TreeViewer

```
public Object[] getElements(Object inputElement) {
         return getChildren(inputElement);
}

public void dispose() {}

public void inputChanged(Viewer viewer, Object
            oldInput, Object newInput) {}
public boolean hasChildren(Object element) {
         return getChildren(element).length > 0;
}
```

Методите dispose() и inputChanged() са с празна имплементация. Тя в случая не е необходима.

Медодът getElements() просто пренасочва своето изпълнение към getChildren(). Пренасочването се извършва за пълнота, защото ITreeContentProvider е наследник на IStructuredContentProvider, а IStructuredContentProvider има метод getElements().

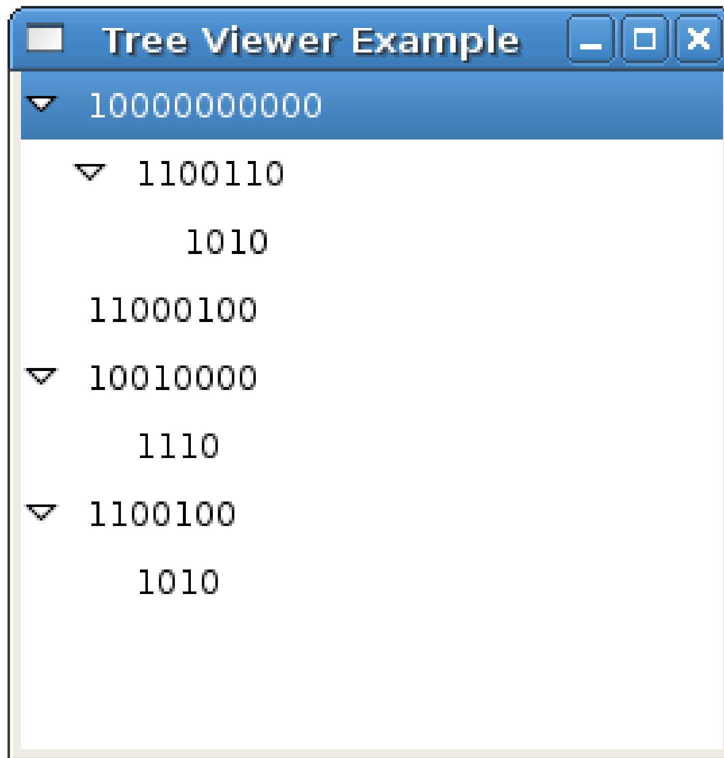
TreeViewer

```
1  public Object[] getChildren(Object parentElement) {
2        if (parentElement instanceof int[]) {
4           int[] array = (int[]) parentElement;
5           Integer[] result = new Integer[array.length];
6           for (int i = 0; i < array.length; i++) {
7              result[i] = new Integer(array[i]);
8           }
9           return result;
10       } else if (parentElement instanceof Integer) {
11          int value = ((Integer)
12                         parentElement).intValue();
13          if (value > 10) return new Integer[] {
               new Integer( value / 10) };
14       }
15       return new Object[0];//must not return null or
}                   //hasChildren() will fail
```

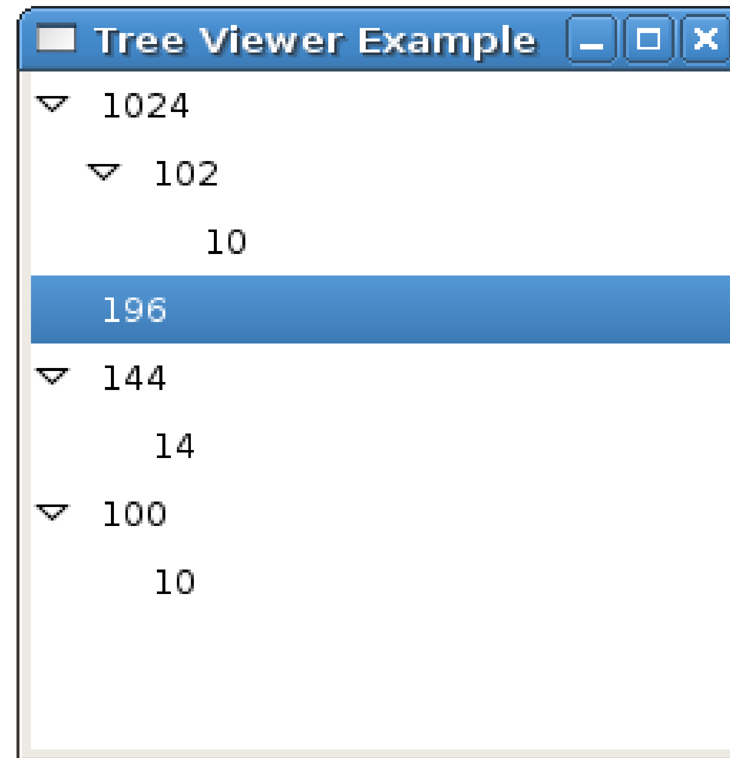
Ред 10-14: **Ако обектът е от тип Integer то се връща масив от един елемент. Този елемент е отново обект от тип Integer, но стойността е вече разделена на десет.**

TreeViewer

Резултатът от изпълнението на програмата е показан на фиг.1. За улеснение на фиг.2 е показан същият резултат, но с десетични числа.



Фиг.1



Фиг.2

Редактиране на таблица

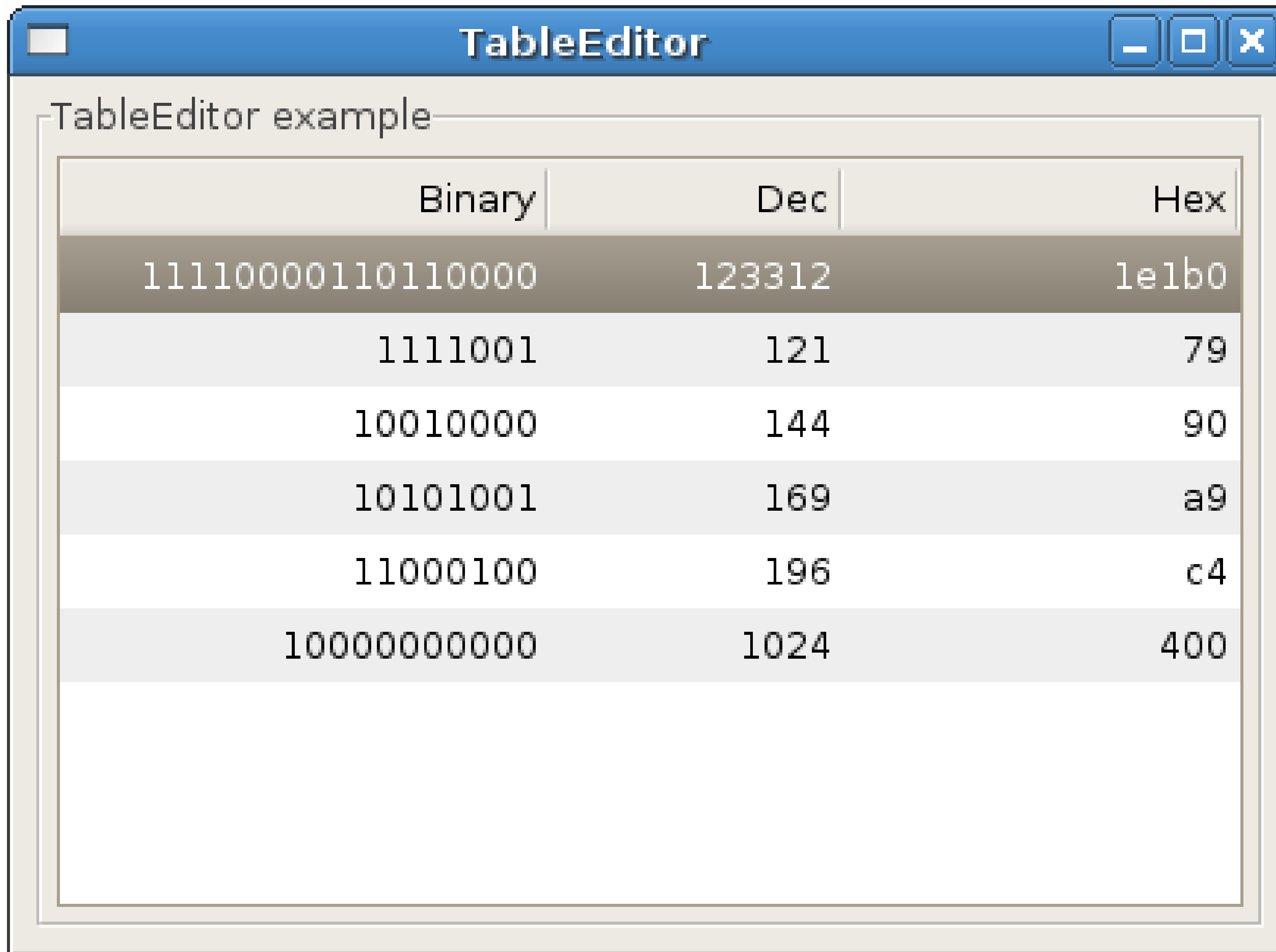
Задачата на този пример е да се изгради TableViewer, който да позволява редактиране на елементите показвани в неговите клетки. Възможно е въвеждането само на цели числа. Позволява се редакция само на средната колона – т.е на тази с десетичните числа.

За целта ще се използва досега разработения TableViewer. С цел премахване на излишната сложност следните класове няма да бъдат използвани:

- **NumbersComparator**
- **NumbersFilter**

TableViewer - Редактиране

Крайният резултат от редактирането ще изглежда по следният начин



TableViewer - Редактиране

Капсулираме необходимите обекти в нов клас - TableEditor

```
1  public class TableEditor {

2      private TableViewer fTableViewer;
3      private int[] fInput = new int[] { 100, 121, 144,
                                              169,196, 1024 };
4      private static String[] COLUMN_NAMES = new String[]
                   { "Binary", "Dec", "Hex" };;
5      public static void main(String[] args) {
6          TableEditor editor = new TableEditor();
7          editor.createComposite();
8      }
   /* ... code missed ...*/
}
```

Ред 2: Обектът от тип TableViewer.

Ред 3: Входните данни за viewer-а.

Ред 4: Имената на колоните на таблицата

TableViewer - Редактиране

Методът TableEditor.createComposite() изглежда по следният начин:

```
public void createComposite() {
/*... code missed - creating shell */
    Group group = new Group(shell, SWT.NONE);
    group.setText("TableEditor example");
    group.setLayout(new GridLayout(1, false));
    group.setLayoutData(new
                    GridData(GridData.FILL_BOTH));
    fTableViewer = createTableViewer(group);
    addColumns(fTableViewer);
    createViewerEditors();
    fTableViewer.setInput(fInput);
/*...code missed – opening shell */
}
```

Основната идея на този метод е да инициализира fTableViewer, добави колоните и създаде обектите необходими за редактиране съдържанието на таблицата. Ще бъда разгледани всичките методи.

TableViewer - Редактиране

```
1 protected TableViewer createTableViewer(Composite
2  parent) {
3  TableViewer viewer = new TableViewer(parent, SWT.BORDER
                                              | SWT.SINGLE);
5  viewer.setContentProvider(new
                        NumbersContentProvider());
6  viewer.setLabelProvider(new
                        NumbersTableLabelProvider());
7  GridData data = new GridData();
8  data.horizontalAlignment = GridData.FILL;
9  data.verticalAlignment = GridData.FILL;
10 data.grabExcessHorizontalSpace = true;
11 data.grabExcessVerticalSpace = true;
12 viewer.getTable().setLayoutData(data);
13 return viewer;
14}
```

Ред 7,8,9,10,11,12: viewer-ът ще бъде разположен, така че да заема цялото му отделено пространство.

TableViewer - Редактиране

```
protected void addColumns(TableViewer viewer) {
      viewer.getTable().setHeaderVisible(true);
      viewer.getTable().setLinesVisible(true);
      for (int i = 0; i < COLUMN_NAMES.length; i++) {
         TableColumn tableColumn = new
            TableColumn(viewer.getTable(), SWT.RIGHT);
         tableColumn.setText(COLUMN_NAMES[i]);
         tableColumn.setWidth(100);
      }
}
```

След създаването на таблицата е необходимо да се създадат и колоните ѝ.

Всяка колона има ИМЕ, ПОДРАВНЯВАНЕ, ШИРИНА.

При създаването си всяка колона приема таблицата като параметър на конструктура си.

TableViewer - Редактиране

Възможността за редактиране изисква използването на следните допълнителни обекти:

- **`org.eclipse.jface.viewers.TextCellEditor` – отговаря за визуалните аспекти относно редактирането и координира работата на останалите обекти свързани с редакцията.**
- **`org.eclipse.jface.viewers.ICellEditorValidator` – позволява валидация на въведената стойност. Разглежданият пример има за цел да позволява въвеждането само на целочислени числа.**
- **`org.eclipse.jface.viewers.ICellModifier` – служи като посредник между входните данни и данните показвани в графичната част. Има за цел да обработи входната информация преди и след редактирането и в графичната част.**

TableViewer - Редактиране

TextCellEditor

```
1   TextCellEditor editor = new
              TextCellEditor(fTableViewer.getTable());
2   fTableViewer.setCellEditors(
              new CellEditor[] { null,editor, null });
```

Ред 1: Създава се нов обект от тип TextCellEditor.

Ред 2: В разработвания пример таблицата има три колони. Следователно на всеки ред има по три клетки. За всяка клетка може да се използва отделен редактор. Това се налага поради факта, че отделните колони/клетки показват различна по тип информация. В една клетка може да се въвеждат низове докато в друга да се избират цветове. В случай целта е да се редактира само средната колона – за другите две не се задават редактори

```
new CellEditor[] { null,editor, null })
```

ICellEditorValidator

```
1   editor.setValidator(new ICellEditorValidator() {
2          public String isValid(Object value) {
3             if (value instanceof String) {
4                try {
5                   Integer.parseInt((String) value);
6                   return null;
7                } catch (NumberFormatException e) {
8                   return "Only numbers allowed";
9                }
10            }
11            return "Unsupported value";
12         }});
```

Ред 2: ICellEditorValidator има един метод – isValid() приемащ като параметър въведената от потребителя стойност. Методът трябва да върне null ако стойността е валидна.
Ред 5: Стойността ще е валидна само ако е текстов низ, който може да се преобразува в цяло число

ICellModifier

Най-сложният обект в дадения пример. Служи като контролер между потребителския интерфейс и входните данни

```
fTableViewer.setCellModifier(new ICellModifier() {
        public boolean canModify(Object element, String property) { /* ...code missed ...*/ }
        public Object getValue(Object element, String property) { /* ...code missed ...*/ }
        public void modify(Object element, String property, Object value) {  /* ...code missed ...*/ }
        }
});
```

И трите метода приемат параметър String property. Целта му е да определи коя колона се редактира. Възможно е да съществуват различни редактори за различните колони, но съществува единствен обект от тип ICellModifier.

TableViewer - Редактиране

ICellModifier

Всяка колона има характеристика отговаряща на параметъра property. При редактирането на дадена колона като параметър се предава тази стойност.

Задаването на property става чрез

```
fTableViewer.setColumnProperties(COLUMN_NAMES);
```

В случай COLUMN_NAMES е масив от низове. Представлява имената на колоните. По този начин имената ще се използват за различаване на различните колони при тяхното редактиране.

TableViewer - Редактиране

ICellModifier - canModify(...)

```
1  fTableViewer.setCellModifier(new ICellModifier() {
2        public boolean canModify(Object element, String
3  property) {
4           if (COLUMN_NAMES[1].equals(property)) {
5              return true;
6           }
7           return false;
8        }
     /* ...code missed...*/
}
```

Методът определя коя колона може да се редактира и коя не. В случай ред 4 проверява дали потребителят се опитва да редактира колоната с характеристика COLUMN_NAMES[1] - "Dec".

TableViewer - Редактиране

ICellModifier - getValue(...)

```
1  fTableViewer.setCellModifier(new ICellModifier() {
2          public Object getValue(Object element, String
3  property) {
4              if (COLUMN_NAMES[1].equals(property)) {
5                  return ((Integer) element).toString();
6              }
7              return null;
8          }
   /* ...code missed ...*/
}
```

Методът определя стойността на дадената колона. В дадения пример задължително трябва да върен низ. Ред 5 извършва преобразуването от обект Integer към String.

TableViewer - Редактиране

ICellModifier - modify(...)

```
1  fTableViewer.setCellModifier(new ICellModifier() {
2     public void modify(Object element, String property,
Object value) {
3        if (element instanceof TableItem
            && value != null) {
4        int index = ((TableItem) element).
                  getParent().
                  indexOf((TableItem) element);
5        if (COLUMN_NAMES[1].equals(property)) {
6        fInput[index] =
               Integer.parseInt((String) value);
7        }
8        fTableViewer.refresh();
      }}
});
```

TableViewer - Редактиране

ICellModifier - modify(...)

```
3          if (element instanceof TableItem
               && value != null) {
4          int index = ((TableItem) element).
                       getParent().
                       indexOf((TableItem) element);
```

Ред 3: Проверка за коректност на предадените данни. Предаденият за редакция елемент трябва да е от тип TableItem. Новата стойност value не трябва да е null. (Възможно е да е null ако се извърши невалидно въвеждане.)

Ред4: TableItem представлява ред от таблицата. В ред 4 намираме индекса на този ред. Този индекс ни е необходим, за да промениме стойността във входния масив. Посочения тук подход се отнася само за разработвания пример. Възможни са други решения.

TableViewer - Редактиране

ICellModifier - modify(...)

```
5           if (COLUMN_NAMES[1].equals(property)) {
6           fInput[index] =
                    Integer.parseInt((String) value);
7           }
8           fTableViewer.refresh();
        }}
```

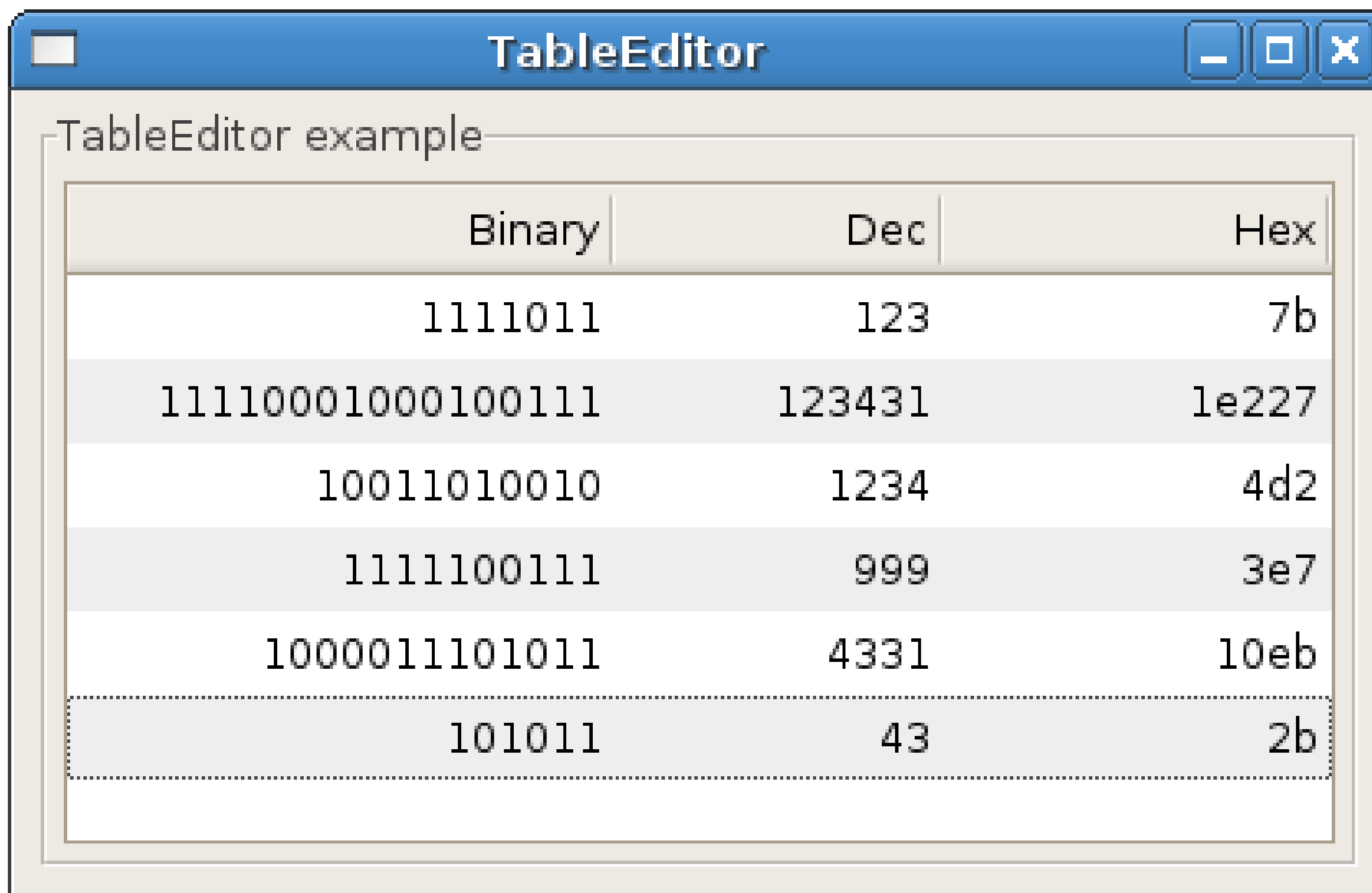
Ред 5: Проверка за колоната.

Ред 6: Нововъведената от потребителя стойност се предава чрез параметъра value. Тъй като вече е намерен индексът на редактирания елемент се променя стойността във входния масив.

Ред 8: Входните данни са променени. Следователно таблицата трябва да се обнови. Обновяването се извършва чрез метода fTableViewer.refresh()

TableViewer - Редактиране

Разработеният viewer дава възможност за редакция и валидация на редактираните стойностти. След промяната на някое число таблицата се обновява и новите стойности се изписват.



Обобщение

- JFace предоставя viewer-и. Тези viewer-и се зползват като обвивка на обектите и улесняват значително тяхното графично изобразяване.
- JFace не предоставя само viewer-и. Налични са още магьосници (wizards), диалогови прозорци (dialogs), класове улесняващи текстообработката и др.
- JFace не е графична библиотека и няма за цел да замени SWT, а да предостави допълнителни улеснения на програмиста при използването на SWT. Възможна е реализацията на графично приложение и без JFace.

Copyright

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 2.5 Bulgaria License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/2.5/bg/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.