

Eclipse

SWT @ TUES

- Java – възможност за платформено-независим код.
- Необходимост и от платформено-независим потребителски интерфейс.
- Първите опити в тази насока са на Sun със създаването на **Abstract Window Toolkit (AWT)**. Предоставя общи компоненти като бутони, списъци и етикети налични на всички платформи, но не предоставя достъп до по-сложни компоненти като таблици и дървета.

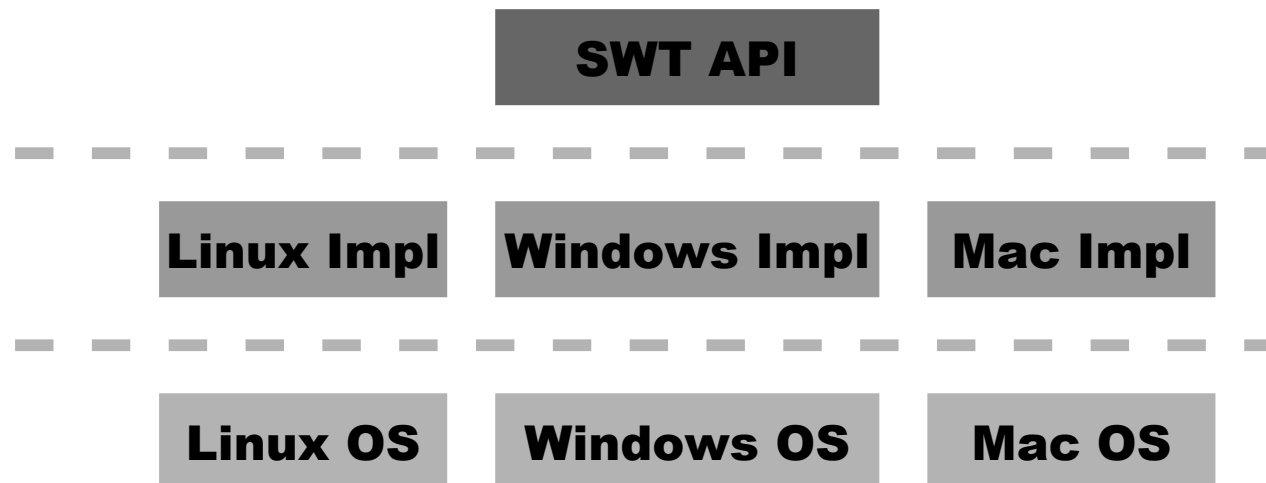
Вторият опит на Sun се нарича **Swing**.

Задачата на тази библиотека за потребителски интерфейс е да реши проблемите на AWT като вместо да се използват компонентите на операционната система се изгражда библиотека симулираща тяхното поведение. Проблемът с малкото компоненти налични в AWT е решен, но остава въпросът с ниската производителност и изгледът на тези приложения.

Swing приложение изглежда като **Swing** приложение не като приложение пуснато на определена операционна система.

SWT - Standard Widget Toolkit

- Библиотека е създадена от IBM.
- Характеризира се с еднослойна структура, където състоянието на компонентите се пази изцяло в компонентите предоставени от операционната система.
- Програмният интерфейс е еднакъв за всички платформи, но директната имплементация е различна за всяка една.



SWT Приложение

```
1  public static void main(String[] args) {
2      Display display = new Display();
3      Shell shell = new Shell(display);
4      shell.setText("Hello World!");
5      shell.setBounds(100, 100, 200, 50);
6      shell.setLayout(new FillLayout());
7      Label label = new Label(shell, SWT.CENTER);
8      label.setText("Hello World!");
9      shell.open();
10     while (!shell.isDisposed()) {
11         if (!display.readAndDispatch())
12             display.sleep();
13     }
14     display.dispose();
15 }
```

(Необходимо е да добавиме SWT библиотеката към проекта)
Кодът може да се изпълни с последователността
Run As -> SWT Application.

Резултат от изпълнението на програмата:



SWT Приложение

```
1  public static void main(String[] args) {
2      Display display = new Display();
3      Shell shell = new Shell(display);
4      shell.setText("Hello World!");
5      shell.setBounds(100, 100, 200, 50);
6      shell.setLayout(new FillLayout());
7      Label label = new Label(shell, SWT.CENTER);
8      label.setText("Hello World!");
9      shell.open();
10     while (!shell.isDisposed()) {
11         if (!display.readAndDispatch())
12             display.sleep();
13     }
14     display.dispose();
15 }
```

Ред 2: Създаване на обект от тип **Display**. **Display** осъществява връзката между платформата и SWT. Предоставя достъп до ресурсите необходими на SWT компонентите.

```
1  public static void main(String[] args) {
2      Display display = new Display();
3      Shell shell = new Shell(display);
4      shell.setText("Hello World!");
5      shell.setBounds(100, 100, 200, 50);
6      shell.setLayout(new FillLayout());
7      Label label = new Label(shell, SWT.CENTER);
8      label.setText("Hello World!");
9      shell.open();
10     while (!shell.isDisposed()) {
11         if (!display.readAndDispatch())
12             display.sleep();
13     }
14     display.dispose();
15 }
```

Ред 3: Обектът от тип **Shell** представлява прозореца създаден от приложението.

Ред 4: Методът **setText()** задава етикет на прозореца.


```
1 public static void main(String[] args) {
2     Display display = new Display();
3     Shell shell = new Shell(display);
4     shell.setText("Hello World!");
5     shell.setBounds(100, 100, 200, 50);
6     shell.setLayout(new FillLayout());
7     Label label = new Label(shell, SWT.CENTER);
8     label.setText("Hello World!");
9     shell.open();
10    while (!shell.isDisposed()) {
11        if (!display.readAndDispatch())
12            display.sleep();
13    }
14    display.dispose();
15 }
```

Ред 5: Методът `setBounds()` задава размерите и положението на прозореца.

Ред 6: Методът `setLayout()` задава менажера отговорен за разположението на обектите.

SWT Приложение

```
1 public static void main(String[] args) {
2     Display display = new Display();
3     Shell shell = new Shell(display);
4     shell.setText("Hello World!");
5     shell.setBounds(100, 100, 200, 50);
6     shell.setLayout(new FillLayout());
7     Label label = new Label(shell, SWT.CENTER);
8     label.setText("Hello World!");
9     shell.open();
10    while (!shell.isDisposed()) {
11        if (!display.readAndDispatch())
12            display.sleep();
13    }
14    display.dispose();
15 }
```

Ред 7,8: Създаване и конфигуриране на нов етикет – обект от тип `org.eclipse.swt.Label`. Етикетът представлява текстово поле без възможност за редактиране. При създаване на всеки SWT компонент е необходимо като параметър на конструкторът да се предаде родителят, в който трябва да се създаде компонента. Всеки компонент може да има единствен родител.

```
1  public static void main(String[] args) {
2      Display display = new Display();
3      Shell shell = new Shell(display);
4      shell.setText("Hello World!");
5      shell.setBounds(100, 100, 200, 50);
6      shell.setLayout(new FillLayout());
7      Label label = new Label(shell, SWT.CENTER);
8      label.setText("Hello World!");
9      shell.open();
10     while (!shell.isDisposed()) {
11         if (!display.readAndDispatch())
12             display.sleep();
13     }
14     display.dispose();
15 }
```

Ред 9: До този момент прозорецът с намиращите се в него компоненти не е видим. С извикването на `shell.open()` се отваря прозорецът.

SWT Приложение

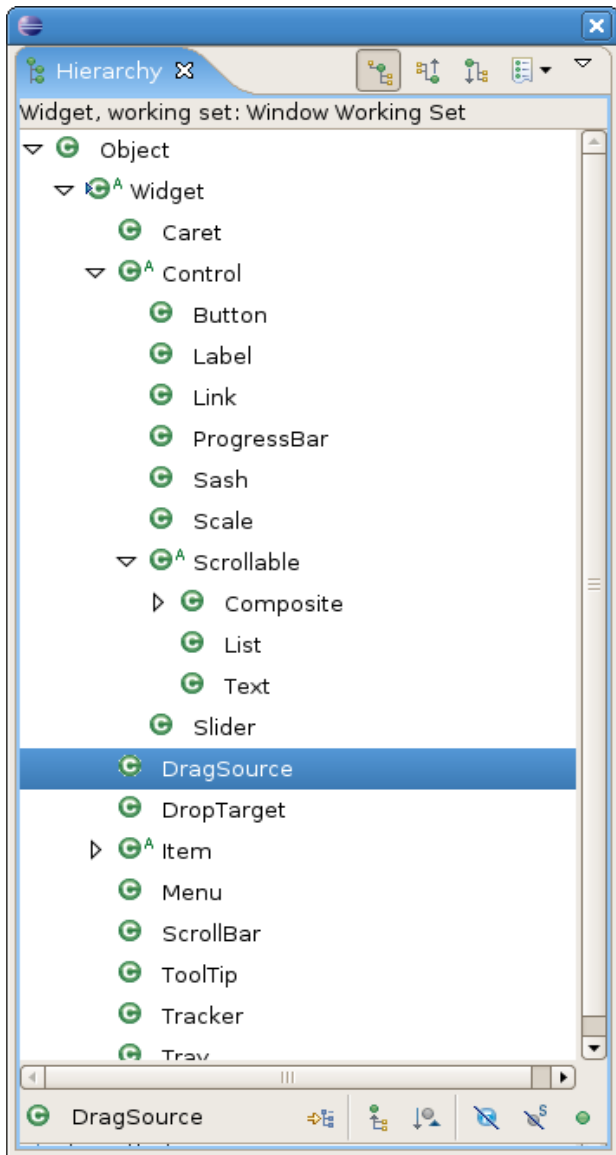
```
1  public static void main(String[] args) {
2      Display display = new Display();
3      Shell shell = new Shell(display);
4      shell.setText("Hello World!");
5      shell.setBounds(100, 100, 200, 50);
6      shell.setLayout(new FillLayout());
7      Label label = new Label(shell, SWT.CENTER);
8      label.setText("Hello World!");
9      shell.open();
10     while (!shell.isDisposed()) {
11         if (!display.readAndDispatch())
12             display.sleep();
13     }
14     display.dispose();
15 }
```

Ред 10,11,12,13: Обектът от тип `Display` има за цел да обработва заявките към графичната система на ОС. Когато такива заявки не се извършват използваната нишка трябва да се освободи. Това се извършва с метода `display.sleep()`. `Display.readAndDispatch()` проверява за налични заявки, а самата проверка се извършва докато `shell.isDisposed()` връща `true`, т.е. докато прозорецът не е затворен.

```
1  public static void main(String[] args) {
2      Display display = new Display();
3      Shell shell = new Shell(display);
4      shell.setText("Hello World!");
5      shell.setBounds(100, 100, 200, 50);
6      shell.setLayout(new FillLayout());
7      Label label = new Label(shell, SWT.CENTER);
8      label.setText("Hello World!");
9      shell.open();
10     while (!shell.isDisposed()) {
11         if (!display.readAndDispatch())
12             display.sleep();
13     }
14     display.dispose();
15 }
```

Ред 14: Освобождаване на използваните от приложението системни ресурси.

Компоненти (Widgets)



- SWT предоставя богат набор от компоненти. Част от йерархията им може да се види на фигурата.
- SWT компонентите пазят колкото се може по-голяма част от тяхното състояние в обекти предоставени от операционната система. *Например текстът съдържащ се в едно текстово поле не се пази като низ от тип `java.lang.String`, а се съдържа изцяло в предоставената от операционната система компонента.*
- SWT компонентите не могат да съществуват самостоятелно, а единствено в определен контекст. След като компонента бъде създадена тя задължително трябва да бъде и разрушена използвайки метода **`Widget.dispose()`**.

Разлика между Window и Linux

- За всяка операционна система се предоставя отделна имплементация.

`org.eclipse.swt.widgets.Display:getSystemFont` имплементиран за Windows

```
public Font getSystemFont () {
    checkDevice ();
    if (systemFont != null) return systemFont;
    int hFont = 0;
    if (!OS.IsWinCE) {
        NONCLIENTMETRICS info = OS.IsUnicode ? (NONCLIENTMETRICS) new
NONCLIENTMETRICSW () : new NONCLIENTMETRICSA ();
        info.cbSize = NONCLIENTMETRICS.sizeof;
        if (OS.SystemParametersInfo (OS.SPI_GETNONCLIENTMETRICS, 0, info,
0)) {
            LOGFONT logFont = OS.IsUnicode ? (LOGFONT)
((NONCLIENTMETRICSW)info).lfMessageFont :
((NONCLIENTMETRICSA)info).lfMessageFont;
            hFont = OS.CreateFontIndirect (logFont);
            lfSystemFont = hFont != 0 ? logFont : null;
        }
    }
    if (hFont == 0) hFont = OS.GetStockObject (OS.DEFAULT_GUI_FONT);
    if (hFont == 0) hFont = OS.GetStockObject (OS.SYSTEM_FONT);
    return systemFont = Font.win32_new (this, hFont);
}
```

Разлика между Window и Linux

- За всяка операционна система се предоставя отделна имплементация.

`org.eclipse.swt.widgets.Display:getSystemFont` имплементиран за Linux

```
public Font getSystemFont () {
    checkDevice ();
    if (systemFont != null) return systemFont;
    int /*long*/ style = OS.gtk_widget_get_style (shellHandle);
    int /*long*/ defaultFont = OS.pango_font_description_copy
(OS.gtk_style_get_font_desc (style));
    return systemFont = Font.gtk_new (this, defaultFont);
}
```


- Пакетите в които се намират компонентите
 - `org.eclipse.swt.widgets`.
 - `org.eclipse.swt.customs`.
- Проектът Nebula - www.eclipse.org/nebula. Supplemental Custom Widgets for SWT (and more). Предоставя нестандартни SWT компоненти.
- Базовият интерфейс за всяка SWT компонента е `org.eclipse.swt.widgets.Widget`

Компонентата Composite

- Използва се като контейнер на други компоненти.
- Децата и се съдържат в нея, изчертават се само в рамките на нейните граници и се оразмеряват при промяна на нейния размер.
- Основна компонента използвана при изграждането на потребителски интерфейс. Автоматично поддържа вертикално и хоризонтално скролиране (при използване на SWT.H_SCROLL и SWT.V_SCROLL като аргумент предаден на конструктора).

```
Composite container = new Composite(shell, SWT.NONE);  
Label label = new Label(container, SWT.NONE);  
Composite childComposite = new Composite(container, SWT.NONE);
```

Компонентата Label

Компонента изобразяваща статичен текст или изображение (няма възможност за редакция от страна на потребителя)

```
Label label = new Label(shell, SWT.CENTER);  
label.setText("Hello World!");
```

```
Label label = new Label(shell, SWT.CENTER);  
Image image = new Image(null, "C:\\Documents and  
Settings\\I041390\\java_course_tues\\linux_penguin_image.jpg");  
label.setImage(image);
```



Компонентата Button

Компонента даваща възможност за създаване на различни видове бутони. Бутоните биват:

```
Button push = new Button(shell, SWT.PUSH);  
push.setText("push");  
Button check = new Button(shell, SWT.CHECK);  
check.setText("check");  
Button radio = new Button(shell, SWT.RADIO);  
radio.setText("radio");  
Button toggle = new Button(shell, SWT.TOGGLE);  
toggle.setText("toggle");  
Button arrow = new Button(shell, SWT.ARROW);  
arrow.setText("arrow");
```



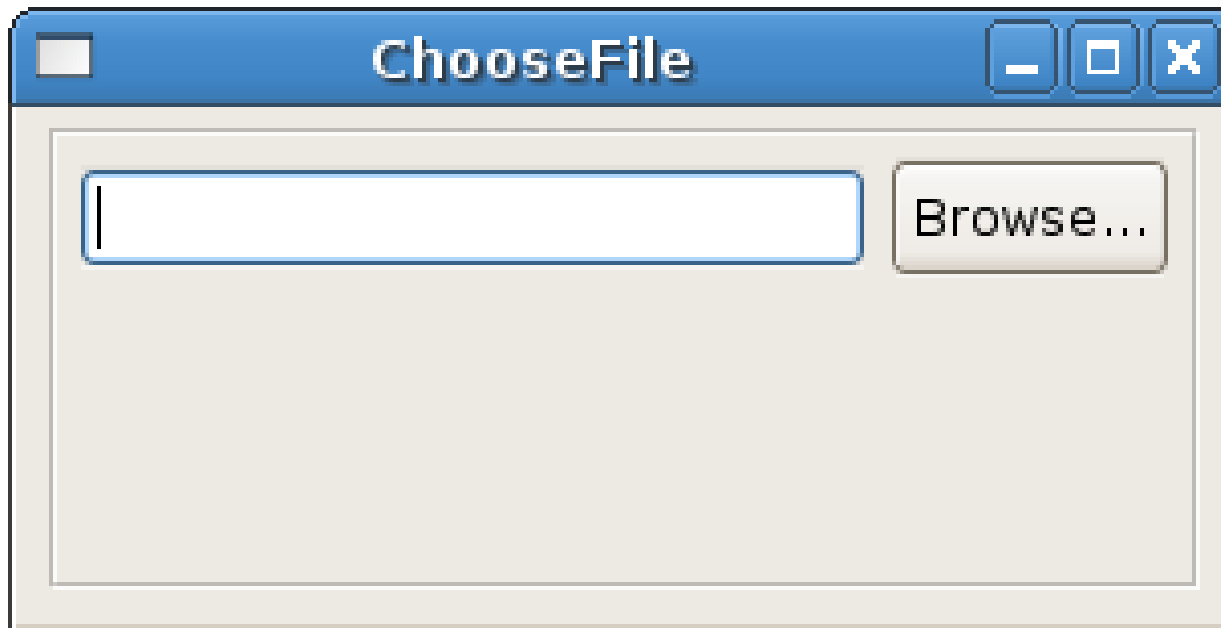
Компонентата Text

Компонента даваща възможност за въвеждане на текст.

```
Text text = new Text(parent, SWT.BORDER);
```

Въведеният от потребителят текст може да се вземе чрез метода `text.getText()`

Началния текст може да се зададе чрез `text.setText()`

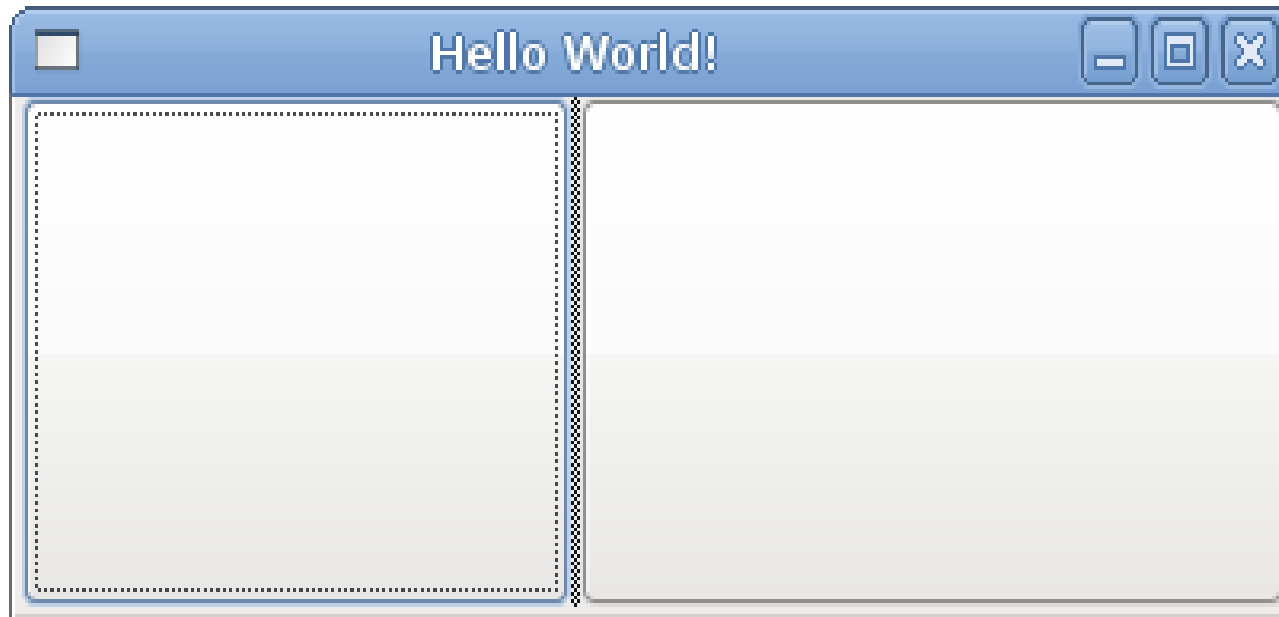


Компонентата SashForm

Компонента от пакета org.eclipse.swt.custom.

```
SashForm form = new SashForm(shell, SWT.HORIZONTAL);  
new Button(form, SWT.PUSH);  
new Button(form, SWT.PUSH);
```

Децата на SashForm се разпределят в две части. Дава се възможност за промяна размера на взяка от двете части.



Стилове

- При създаването на всяка SWT компонента може да се определи нейният стил. Стилът определя изгледа и поведението на компонентата.
- Константите използвани за определяне стила се намират в `org.eclipse.swt.SWT`
- Често използвани стилове
`SWT.NONE`, `SWT.PUSH`, `SWT.BORDER`, `SWT.DEFAULT`
- Позволява се групиране на стиловете
`SWT.V_SCROLL` | `SWT.H_SCROLL`

Стилове

- Стиловете поддържани от всяка компонента са описани в документацията на конструктора ѝ.

```
*
* @see SWT#ARROW
* @see SWT#CHECK
* @see SWT#PUSH
* @see SWT#RADIO
* @see SWT#TOGGLE
* @see SWT#FLAT
* @see SWT#LEFT
* @see SWT#RIGHT
* @see SWT#CENTER
* @see Widget#checkSubclass
* @see Widget#getStyle
*/
public Button (Composite parent, int style) {
    super (parent, checkStyle (style));
}
```


- Събитията са предназначени да съобщават на приложението за извършените от потребителя действия.
- Например при натискане на бутон, изписване на текст, движение на мишката и т.н.
- За всяко събитие в SWT е дефиниран съответен интерфейс. Всяка SWT компонента поддържа определени събития.
- Нотификацията се извършва с помощта на предварително дефиниран “*listener*” (наблюдател). Наблюдателят се регистрира в компонентата и получава съобщенията, за които се е регистрирал, при тяхното настъпване.
- Често използвани събития:
 - **Selection** - Обектът е селектиран
 - **Modify** – Обектът е модифициран
 - **Mouse Move** – Мишката влиза, застава върху или излиза от рамките на компонентата

SelectionEvent

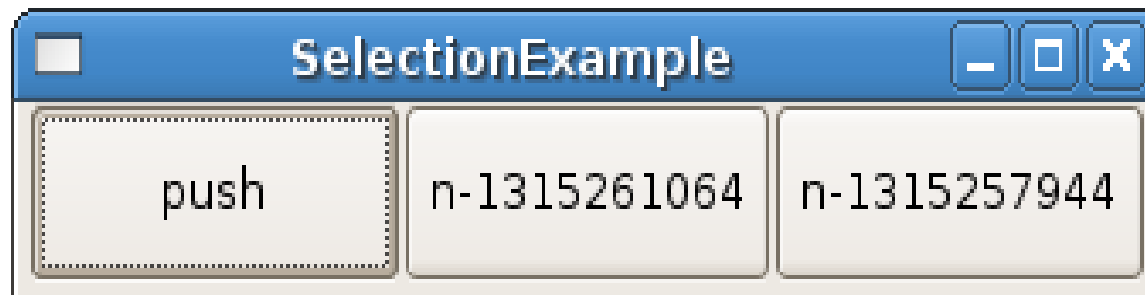
За да бъдем оповестени за събитие от този тип е необходимо да сме регистрирали обект от тип **SelectionListener** в компонентата.

```
final Shell shell = new Shell(display);
shell.setText("SelectionExample");
shell.setBounds(100, 100, 200, 150);
shell.setLayout(new FillLayout());
Button push = new Button(shell, SWT.PUSH);
push.setText("push");
push.addSelectionListener(new SelectionListener() {
    public void widgetDefaultSelected(SelectionEvent e){
    }
    public void widgetSelected(SelectionEvent e) {
        Button nextButton = new Button(shell, SWT.PUSH);
        nextButton.setText("n" + e.time);
        shell.layout(true);
    }
});
```

При селектиране на компонентата се извиква методът

```
widgetSelected(SelectionEvent e){...}
```

SelectionEvent



В случая `org.eclipse.swt.events.SelectionListener` е интерфейс.

Някой от интерфейсите от тип `*Listener` имат имплементации с поведение по подразбиране. Тези имплементации са класове с име `*Adapter`.

В случая на `org.eclipse.swt.events.SelectionListener` имплементацията е `org.eclipse.swt.events.SelectionAdapter`.

Използването на `*Adapter` класовете може да доведе до по ясен и чист код.

Менажери (LayoutManager)

- SWT компонента се създава в даден родител, съществува в границите на родителя си, и се унищожава при унищожаване на родителя си.
- Тъй като един родител може да има повече от едно дете е необходима стратегия за позиционирането на децата в рамките на родителя. Стратегията трябва да позволява динамично пренареждане на компонентите.
- Позиционирането се извършва чрез използването на “менажер”. Съществуват различни видове менажери. Задаването му става чрез използване на метода `setLayout(Object layout)`.
- Съществуват 4 вида менажери
 - `org.eclipse.swt.layout.FillLayout`
 - `org.eclipse.swt.layout.RawLayout`
 - `org.eclipse.swt.layout.GridLayout`
 - `org.eclipse.swt.layout.FormLayout`

Липса на менажери

Когато не е зададена стратегия то позиционирането се извършва изцяло на база координатите на компонентите деца (по подразбиране децата се намират на координати 0,0 и имат размери 0,0).

```
final Shell shell = new Shell(display);
shell.setText("ModifyEvent");
shell.setBounds(100, 100, 200, 100);
final Button b1 = new Button(shell, SWT.PUSH);
b1.setText("b1");
b1.setBounds(10, 10, 45, 45);
final Button b2 = new Button(shell, SWT.PUSH);
b2.setText("b2");
b2.setBounds(30, 20, 70, 40);
```



FillLayout

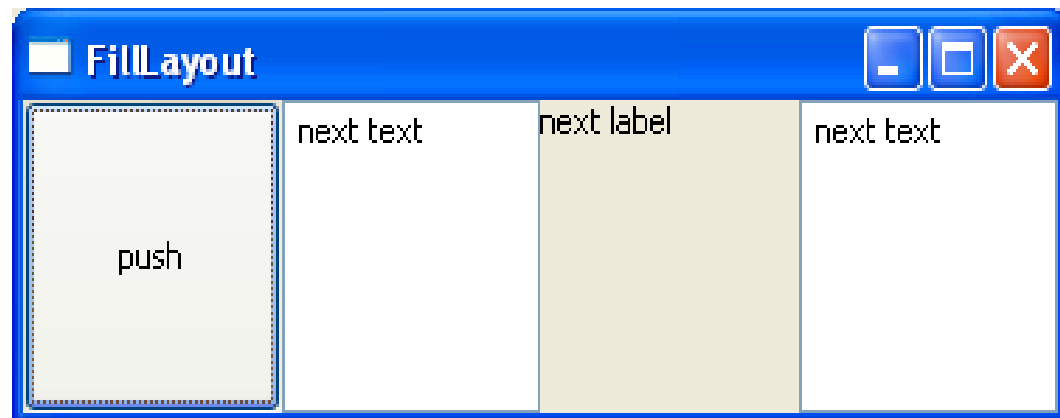
Компонентите деца се разполагат така, че максимално да запълнят пространството предоставено от родителя.

```
final Shell shell = new Shell(display);
shell.setText("FillLayout");
shell.setBounds(100, 100, 200, 150);
shell.setLayout(new FillLayout(SWT.V_SCROLL)); //H_SCROLL
final Button push = new Button(shell, SWT.PUSH);
push.setText("push");
push.addSelectionListener(new SelectionAdapter() {
    boolean fIsLabel = true;
    public void widgetSelected(SelectionEvent e) {
        fIsLabel = !fIsLabel;
        if (fIsLabel) {
            Label label = new Label(shell, SWT.NONE);
            label.setText("next label");
        } else {
            Text text = new Text(shell, SWT.BORDER);
            text.setText("next text");
        }
        shell.layout();
    }
});
```



- Вертикално разположение

- Хоризонтално разположение



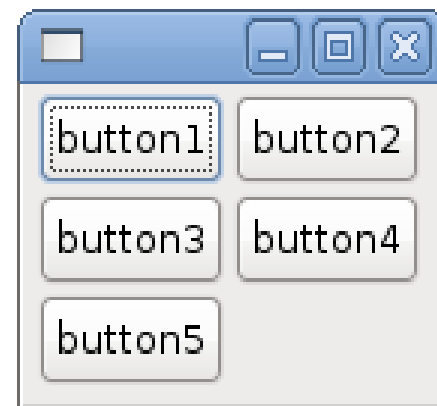
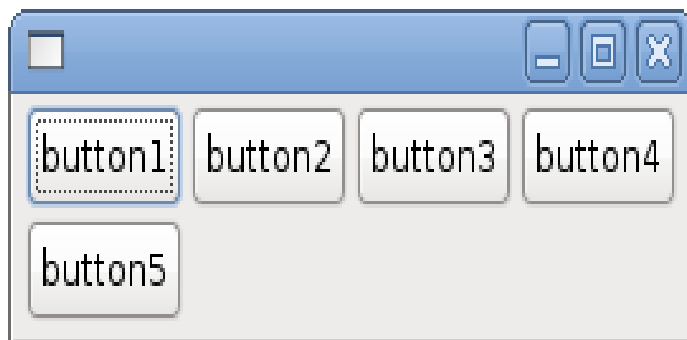
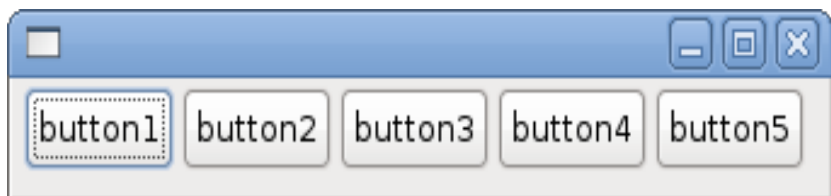
RowLayout

Компонентите деца се разполагат в ред. Редът може да бъде хоризонтален или вертикален. Целта е всички компоненти да се разположат в един ред. Ако това не е възможно се създават нови редове в които последователно се разполагат компонентите.

```
Shell shell = new Shell();
RowLayout layout = new RowLayout();
shell.setLayout(layout);
Button button1 = new Button(shell, SWT.PUSH);
button1.setText("button");
Button button2 = new Button(shell, SWT.PUSH);
button2.setText("button");
Button button3 = new Button(shell, SWT.PUSH);
button3.setText("button");
Button button4 = new Button(shell, SWT.PUSH);
button4.setText("button");
Button button5 = new Button(shell, SWT.PUSH);
```


RowLayout

При промяна размера на родителя децата променят разположинето си така, че да се намират в редове.

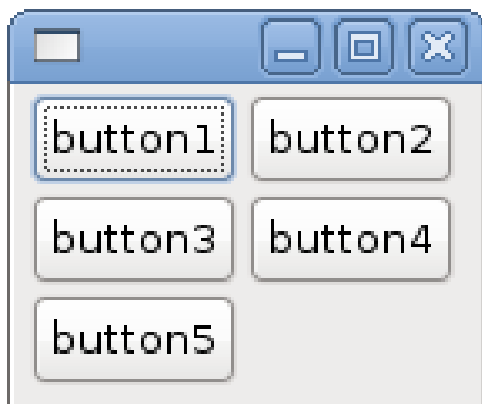


RowLayout

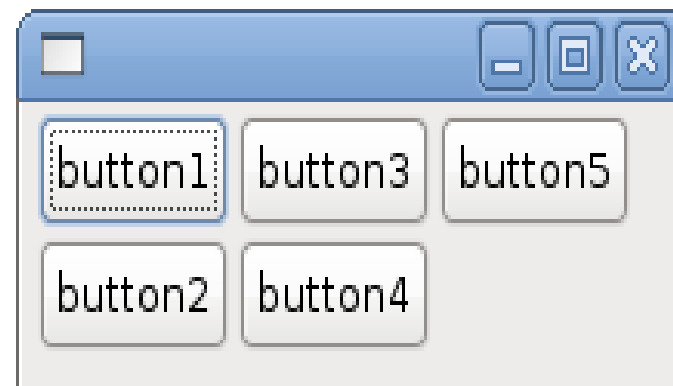
Компонентите може да се разположат във вертикални или хоризонтални колони/редове.

Стилът се посочва чрез

```
RowLayout layout = new RowLayout();  
layout.type = SWT.HORIZONTAL;  
layout.type = SWT.VERTICAL;
```



■ Редове



■ Колони

FormLayout

Компонентите деца се разполагат една спрямо друга. За да се разположи дадена компонента се посочват какво е нейното отношение спрямо родителя и, или спрямо компонентите около нея.

За работа с `FormLayout` се използват класовете `FormLayout`, `FormData`, `FormAttachment`.

На родителя се задава `FormLayout`.

На всяко дете може да се зададе обект от тип `FormData`.

`FormData` съдържа четири обекта от тип `FormAttachment`.

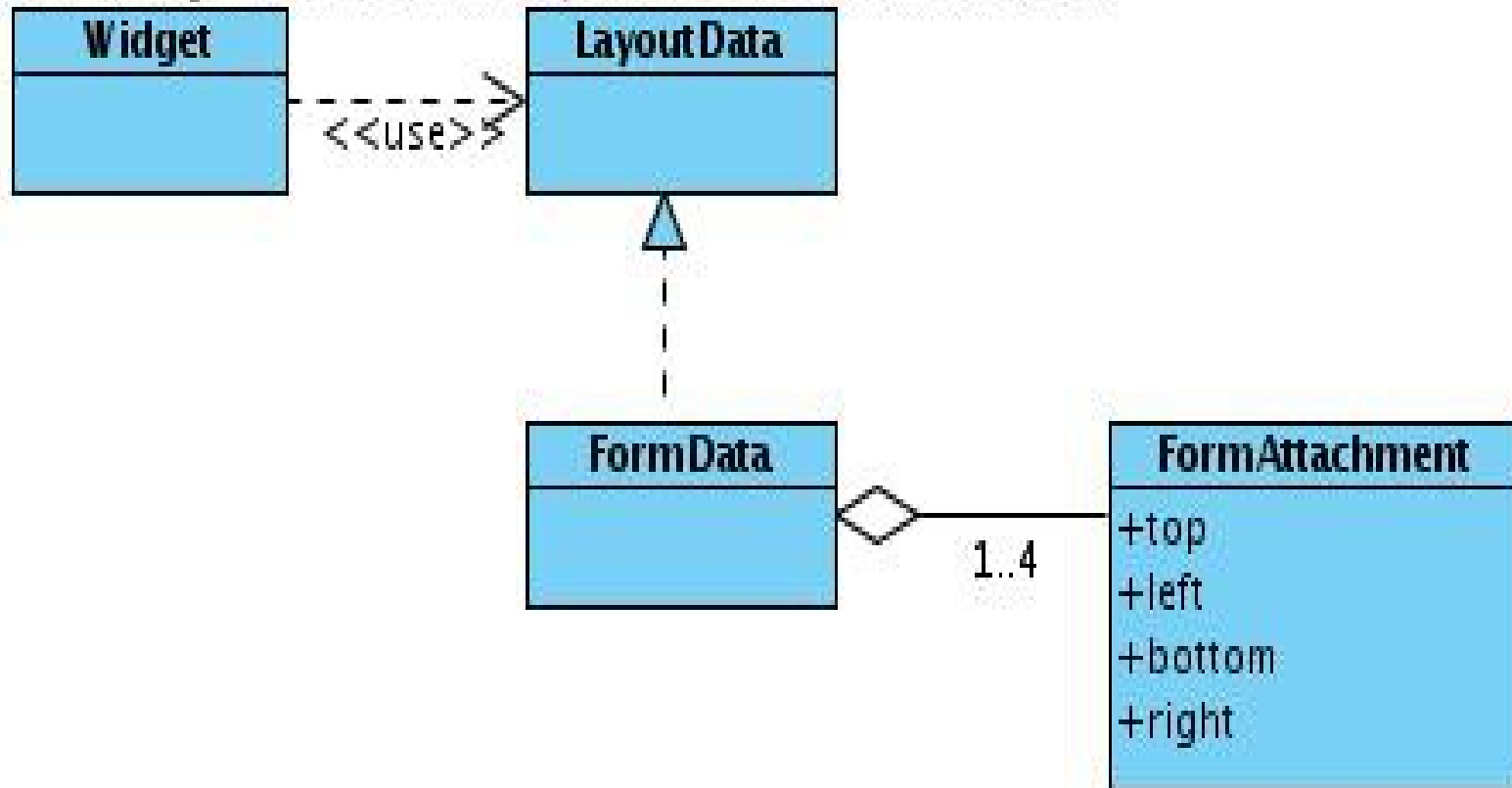
Всеки компонент има четири страни – `top`, `bottom`, `left`, `right`.

Един `FormAttachment` показва как се държи дадена страна от компоненти спрямо другите компоненти.

Затова всяка `FormData` съдържа четири `FormAttachment`

FormLayout

Visual Paradigm for UML Community Edition [not for commercial use]



FormLayout

Когато е зададен само `FormLayout` и не се използват `FormData` и `FormAttachment` компонентите се поставят една върху друга.

```
Shell shell = new Shell(display);  
shell.setLayout(new FormLayout());  
new Button(shell, SWT.PUSH).setText("button1");  
new Button(shell, SWT.PUSH).setText("button2");
```



Зад `button1` се намира `button2`

FormLayout

Компонентът **button2** се закача/позиционира едновременно спрямо родителя се и спрямо **button**

```
Shell shell = new Shell(display);
FormLayout layout = new FormLayout();
shell.setLayout(layout);
Button button = new Button(shell, SWT.PUSH);
button.setText("button1");
Button button2 = new Button(shell, SWT.PUSH);
button2.setText("button2");
FormData data = new FormData();
button2.setLayoutData(data);
data.bottom = new FormAttachment(100, 0);
data.top = new FormAttachment(button, 5);
data.left = new FormAttachment(button, 0, SWT.LEFT);
data.right = new FormAttachment(button, 0, SWT.RIGHT);
```

FormLayout

Долната част на **button2** – **bottom**, ще се оразмерява на 100 % спрямо долната част на родителя си, като отместването ще бъде 0.

```
data.bottom = new FormAttachment(100, 0);
```

Дгорната част на **button2** ще се оразмерява и позиционира спрямо **button**, но ще се постави отместване 5.

```
data.top = new FormAttachment(button, 5);
```

Лявата и дясната страна на **button2** ще се подравнят с лявата и дясната страна на **button**. Отместването ще бъде 0

```
data.left = new FormAttachment(button, 0, SWT.LEFT);  
data.right = new FormAttachment(button, 0, SWT.RIGHT);
```

FormLayout

Показани са състоянието преди и след оразмеряване



FormLayout

FormLayout е най-мощният стандартен SWT layout manager. Теоретично е възможно всеки ефект постигнат с друг layout manager да се постигне с **FormLayout**. Но **FormLayout** предоставя много повече. Предоставя възможности, които не могат да бъдат предоставени от другите Layout менажери.

FormLayout е по-рядко използван поради неговата допълнителна сложност. Голяма част от ефектите желани в позиционирането на компонентите може да бъдат постигнати по-лесно чрез използване на **GridLayout**.

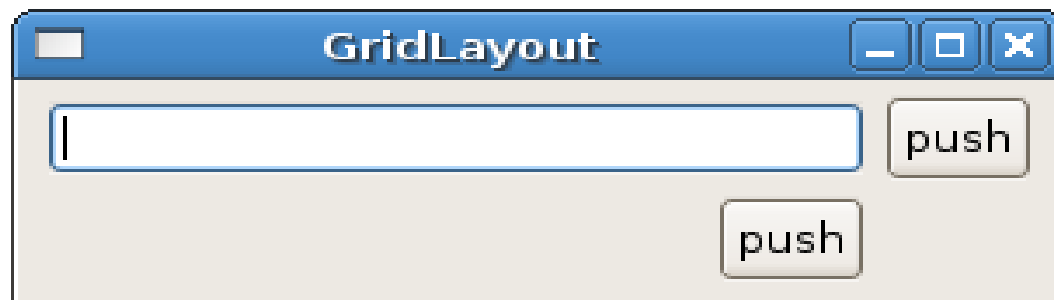
- Можеби най-често използвания менажер. Предоставя изключително много възможности.
- Децата се разполагат във вид на таблица. Всяко дете се поставя в собствена клетка. Детето може да заема различни положения в хоризонтална/ вертикална посока в клетката:
 - SWT.*END* – крайно дясно
 - SWT.*BEGINNING* – крайно ляво
 - SWT.*CENTER* – по средата
- Менажерът използва информация предоставена от детето при неговото разполагане. При *GridLayout* тази информация се предоставя чрез обект от тип *org.eclipse.swt.layout.GridData*.
- Всяко дете трябва да има свой собствен обект от тип *GridData* – не се допуска един и същи обект да бъде зададен на две или повече компоненти.
- Задаването на *GridData* става чрез метода *setLayoutData(Object data)*.

GridLayout

```
1 final Shell shell = new Shell(display);
2 shell.setText("GridLayout");
3 shell.setBounds(100, 100, 300, 100);
4 shell.setLayout(new GridLayout(2, false));
5 final Text text1 = new Text(shell, SWT.BORDER);
6 GridData data = new GridData();
7 data.horizontalAlignment = SWT.FILL;
8 data.grabExcessHorizontalSpace = true;
9 text1.setLayoutData(data);
10 final Button centerButton = new Button(shell,
SWT.PUSH);
11 centerButton.setText("push");
12 centerButton.setLayoutData(new GridData(SWT.CENTER));
13 final Button rightButton = new Button(shell,
SWT.PUSH);
14 rightButton.setText("push");
15 data = new GridData();
16 data.horizontalAlignment = SWT.END;
17 rightButton.setLayoutData(data);
```

```
4 shell.setLayout(new GridLayout(2, false));
```

Ред 4: Обектът от тип **Shell** ще използва **GridLayout**, с 2 колони които може да са различни по ширина, за да разполага децата си.



GridLayout

```
6 GridData data = new GridData();  
7 data.horizontalAlignment = SWT.FILL;  
8 data.grabExcessHorizontalSpace = true;
```

Ред 6, 7, 8: Създава се обект от тип **GridData**. Посочва се хоризонталното разположение. Ако след разполагане на компонентата е налично допълнително място в хоризонтална посока то детето се разширява. При промяна на Ред 8: до `data.grabExcessHorizontalSpace = false;`

то разположението е следното:



Ресурси на операционната система

- Всяка операционна система предоставя определен набор от цветове, шрифтове и изображения.
- Операционните системи налагат ограничения върху броя на обектите (цветове, шрифтове и изображения), които могат да съществуват едновременно в системата.
- Изключително важно е правилното управление на подобни ресурси от страна на разработчика.
- Основното правило е „Ако създадете ресурс, то при приключване работата с него трябва да го освободите. Ако не вие сте създали ресурса не трябва да го унищожавате“

Шрифтове

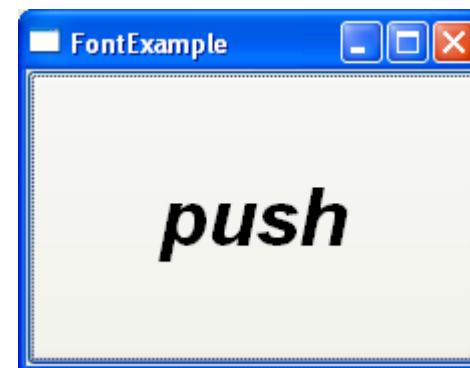
- Всеки шрифт се характеризира с име („Arial“, „Monospace“, т.н), стил (*SWT.BOLD*, *SWT.ITALIC*, *SWT.NORMAL*) и големина.

```
Font font = new Font(display, "Monospace", 30,  
                      SWT.BOLD | SWT.ITALIC);
```

създава шрифт с име „Monospace“, големина 30, който е едновременно *BOLD* и *ITALIC*.

- Шрифтът се задава с помоща на метода `setFont(Font font)`
- Когато шрифтът не е вече необходим трябва да се извика `font.dispose()`;

което освобождава системните ресурси.



- Подобно на шрифтовете цветовете са ограничен системен ресурс. Създаването на цвят изглежда по следният начин:

```
Color color1 = new Color(null, 0/* red */, 255/* green  
*/, 0/* blue */);
```
- Класът *org.eclipse.swt.widgets.Display* предоставя готови обекти от тип *Color*.

```
Color color2 =  
Display.getCurrent().getSystemColor(SWT.COLOR_RED);
```
- Задължително е да се извика `color1.dispose()` и да не се извика `color2.dispose()`.



Изображения

- Поддържат се голям набор от формати (JPG, GIF, PNG, т.н.).
- Изображенията в SWT са обекти от тип `org.eclipse.swt.graphics.Image`.
- Създаване на изображение

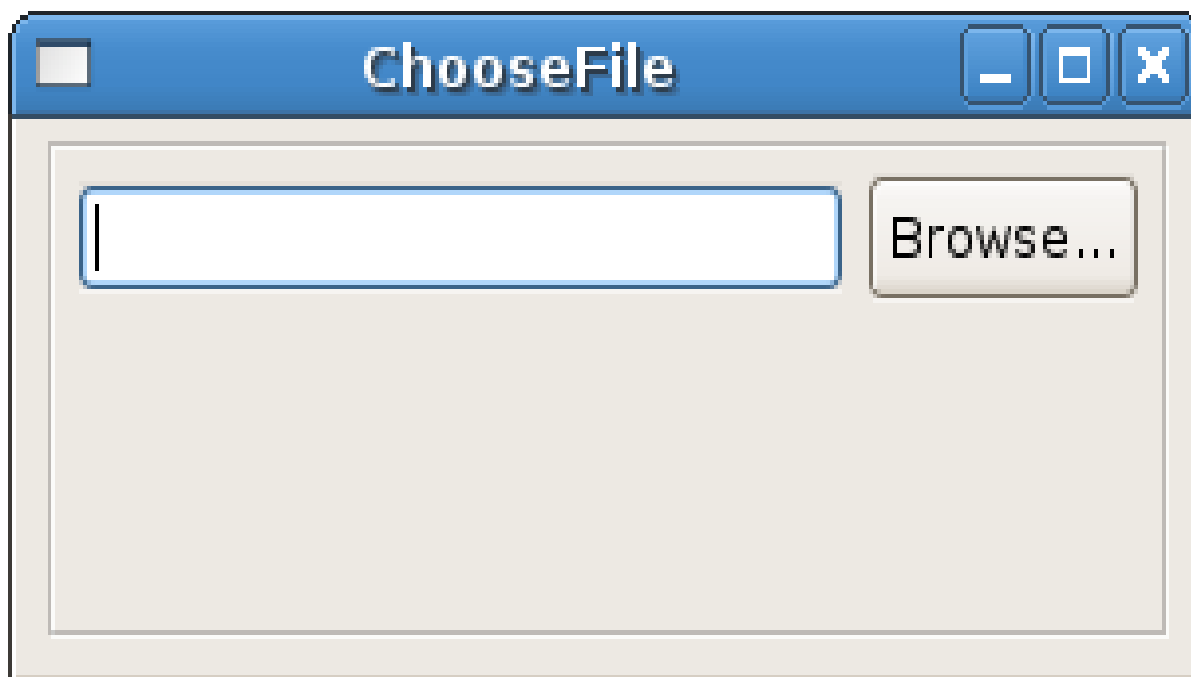
```
Image image = new Image(null,  
                        "/home/kiril/myImage.jpg");
```



Пример

Да се изгради визуална компонента позволяваща избор на файл от файловата система.

Файлът да може да се избере чрез диалогов прозорец или да се напише пътят към него в текстово поле.



Пример

Клас капсулиращ необходимите елементи

```
public class ChooseFileExample {  
    private String fileName;  
    private Text textField;  
    private Button browseButton;  
    public String getFileName() {  
        return fileName;  
    }  
    /* ... */  
}
```

Пример

За да се избегне статичният подход капсулираме функционалността в метода `createComposite()`

```
public static void main(String[] args) {
    ChooseFileExample example = new ChooseFileExample();
    example.createComposite();
}
```

```
public void createComposite() {
    /*...code missed ...*/
    shell.setLayout(new GridLayout(1, false));
    Group group = new Group(shell, SWT.NONE);
    group.setLayout(new GridLayout(2, false));
    group.setLayoutData(new
        GridData(GridData.FILL_BOTH));
    fTextField = createTextField(group);
    fBrowseButton = createBrowseButton(group);
    addButtonListener();
    addTextListener();
    /*...code missed ...*/
}
```

Текстово поле

Създаването на текстовото поле и на бутона се извършва в отделни методи. Този подход позволява по-лесна наследяване и тестване на дадения клас.

```
1  protected Text createTextField(Composite parent) {
2      Text text = new Text(parent, SWT.BORDER);
3      GridData data = new GridData();
4      data.horizontalAlignment = GridData.FILL;
5      data.grabExcessHorizontalSpace = true;
6      text.setLayoutData(data);
7      return text;
8  }
```

От изключителна важност са редове 3, 4, 5, 6 тъй като те позволяват на текстовото поле да променя размера си в зависимост от големината на прозореца

Текстово поле - събития

При редактиране на текстовото поле е необходимо да промениме променливата **fFileName**. За това спомага наблюдателят, който ще бъде добавен.

```
1  protected void addTextListener() {
2      fTextField.addModifyListener(new ModifyListener() {
3          public void modifyText(ModifyEvent e) {
4              fFileName = fTextField.getText();
5          }
6      });
7  }
```

Използва се обект от тип **ModifyListener**. При всяка промяна на текстовото поле ще бъде променяно и полето **fFileName**

Бутон

Подобно на текстовото поле и бутонът се създава в отделен метод.

```
protected Button createBrowseButton(Composite parent) {  
    Button browse = new Button(parent, SWT.PUSH);  
    browse.setLayoutData(new GridData());  
    browse.setText("Browse...");  
    return browse;  
}
```

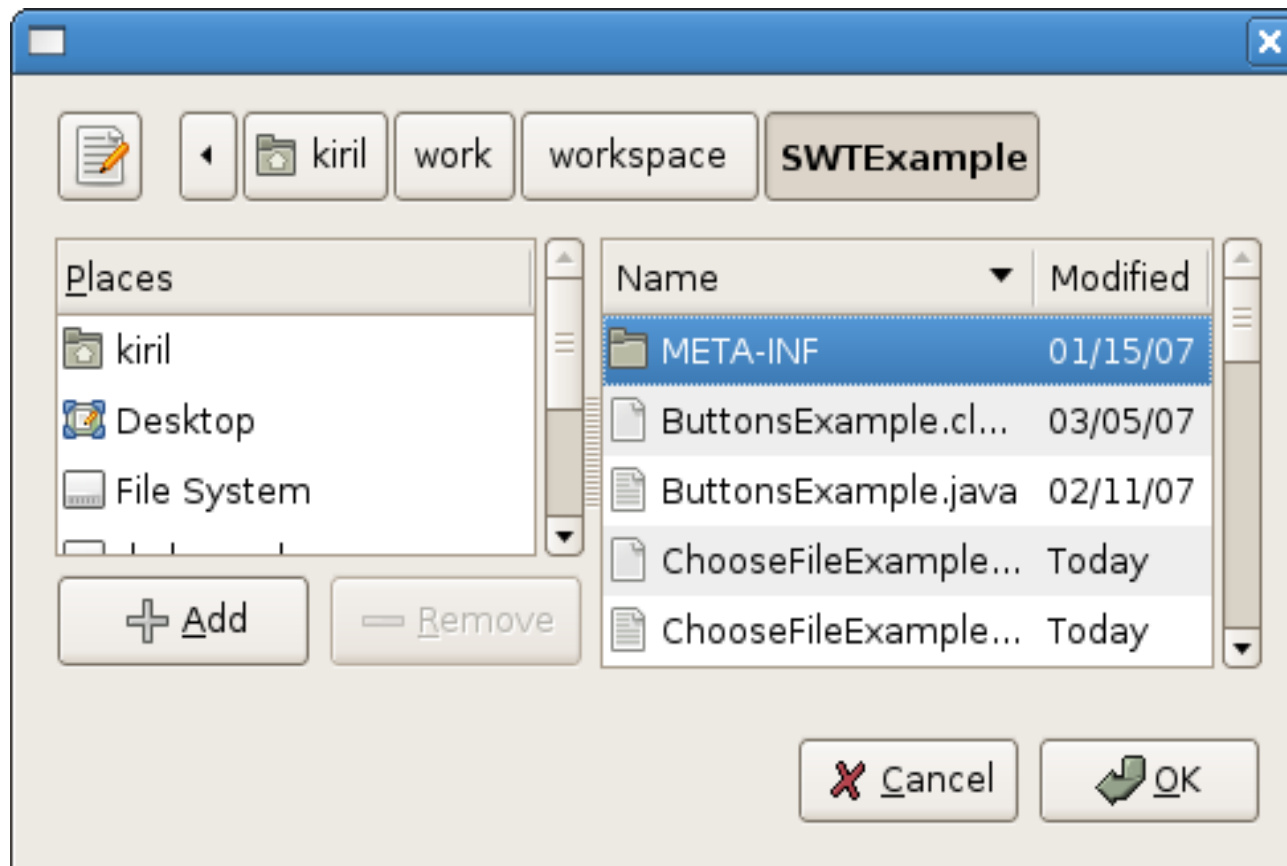
Бутон - събития

При селектирането на бутон трябва да се отвори диалогов прозорец за избор на файл от файловата система. За целта може да се използва класът `org.eclipse.swt.widgets.FileDialog`

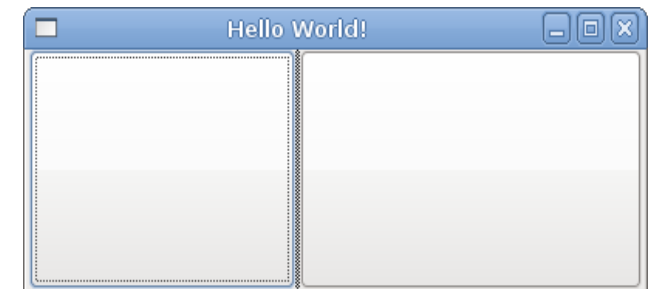
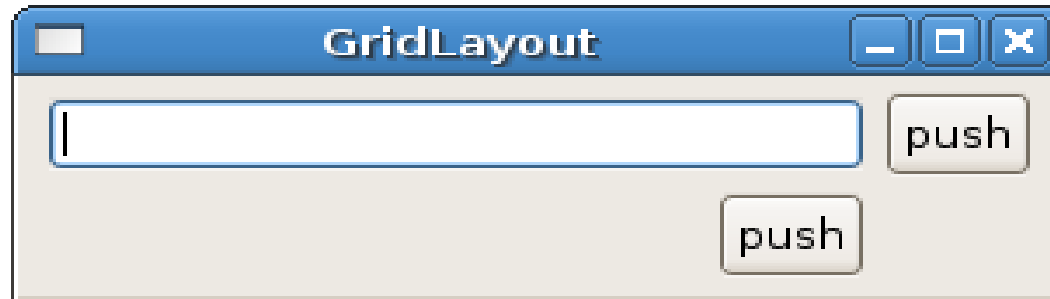
```
protected void addButtonListener() {
    fBrowseButton.addSelectionListener(new
        SelectionAdapter() {
            @Override
            public void widgetSelected(SelectionEvent e) {
                FileDialog fileDialog = new
                    FileDialog(fBrowseButton.getShell());
                if (fileDialog.open() != null) {
                    fFileName = fileDialog.getFileName();
                    fTextField.setText(fFileName);
                }
            }
        });
}
```


Бутон - събития

Методът `FileDialog.open()` връща името на избрания файл или `null` ако е натиснат `Cancel`

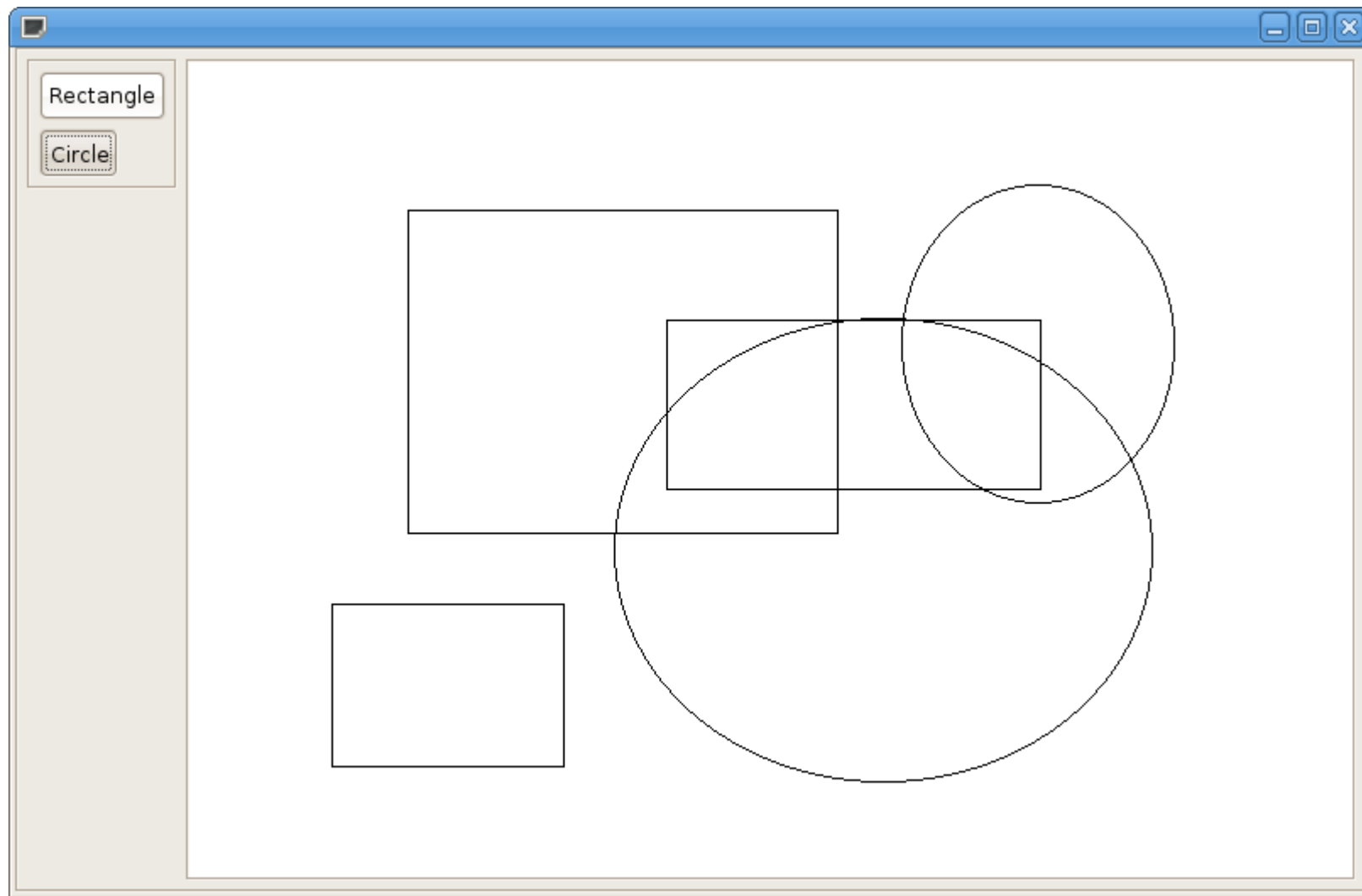


SWT разполага с много компоненти:



Canvas

Canvas е SWT компонента предоставяща основа при изчертаването на произволна графика.

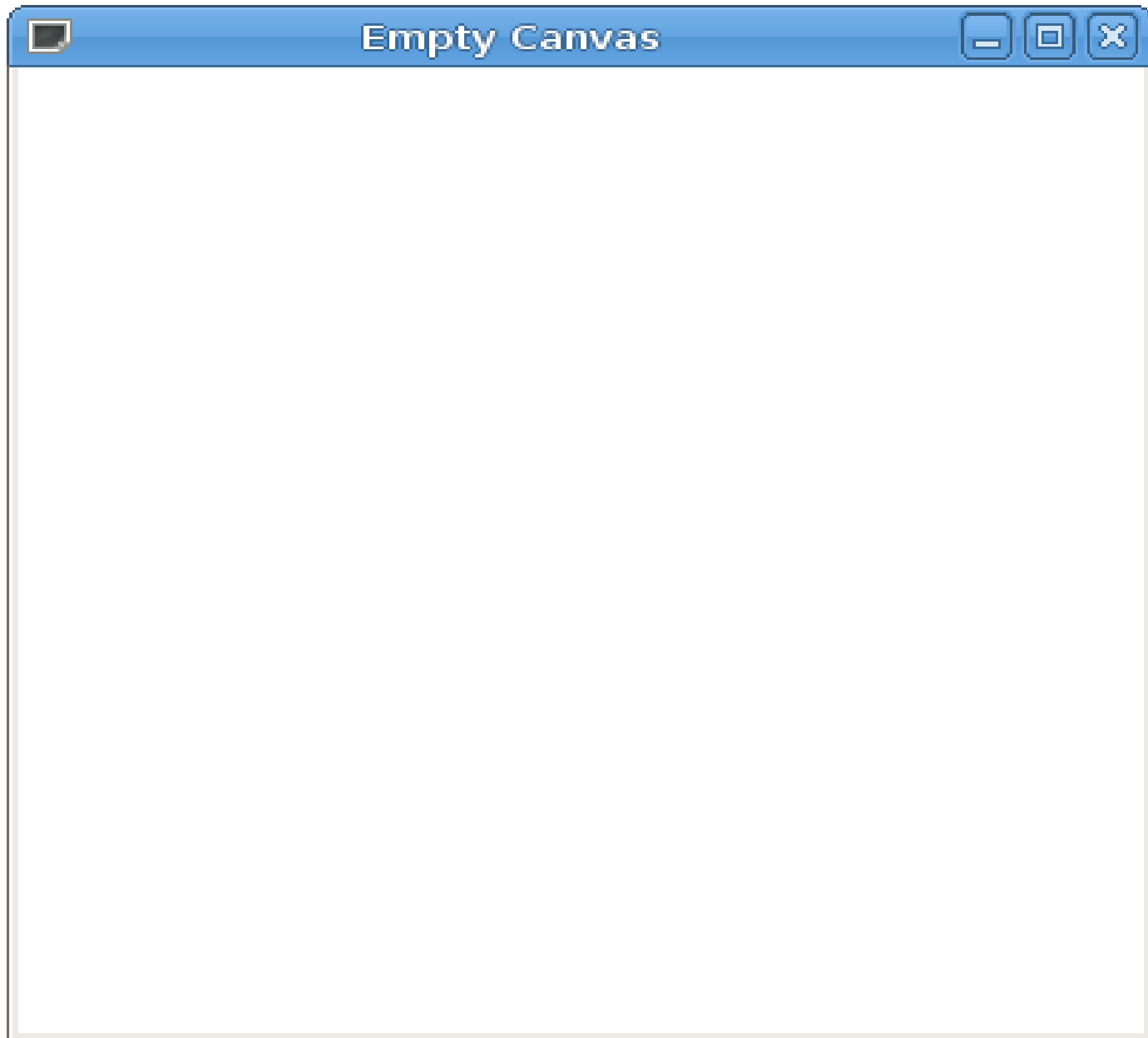


SWT Приложение с Canvas

```
1 public static void main(String[] args) {
2     Display display = new Display();
3     Shell shell = new Shell(display);
4     shell.setText("Empty Canvas");
5     shell.setBounds(100, 100, 400, 400);
6     shell.setLayout(new FillLayout());
7     Canvas canvas = new Canvas(shell, SWT.NONE);
8     canvas.setBackground(Display.getDefault().getSystemColor(
9         SWT.COLOR_WHITE));
10    shell.open();
11    while (!shell.isDisposed()) {
12        if (!display.readAndDispatch())
13            display.sleep();
14    }
15    display.dispose();
16 }
```

Ред 7-9: Създаване на обект от тип **Canvas** служещ като основа при изчертаване на графика. Задаване цвета на задния фон на основата.

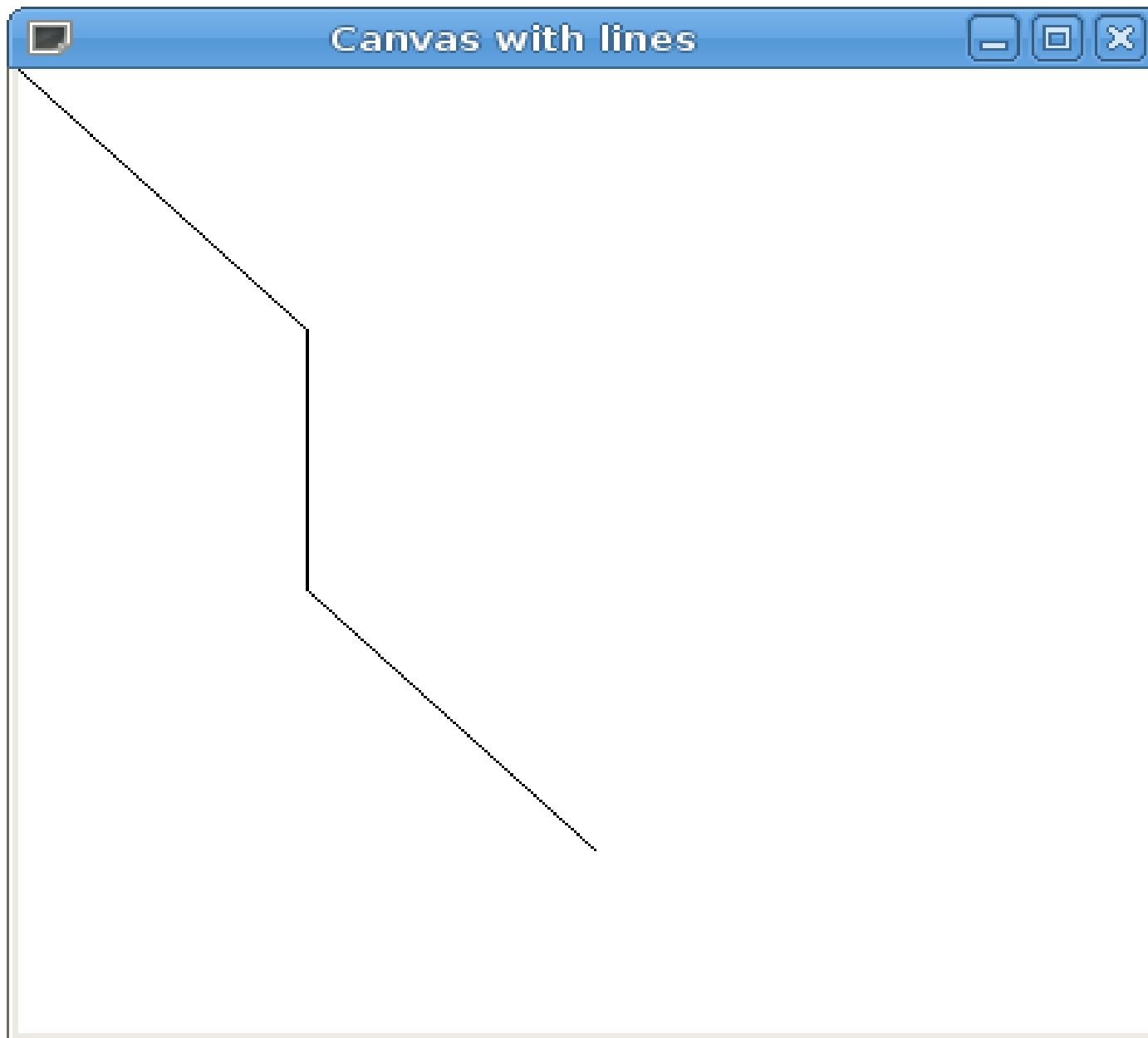
SWT Приложение с Canvas



Изчертаването върху полето става с помощта на обект от тип **GC** (**Graphics Context**)

```
GC gc = new GC(canvas);  
gc.drawLine(0, 0, 100, 100);  
gc.drawLine(100, 100, 100, 200);  
gc.drawLine(100, 200, 200, 300);  
gc.dispose();
```

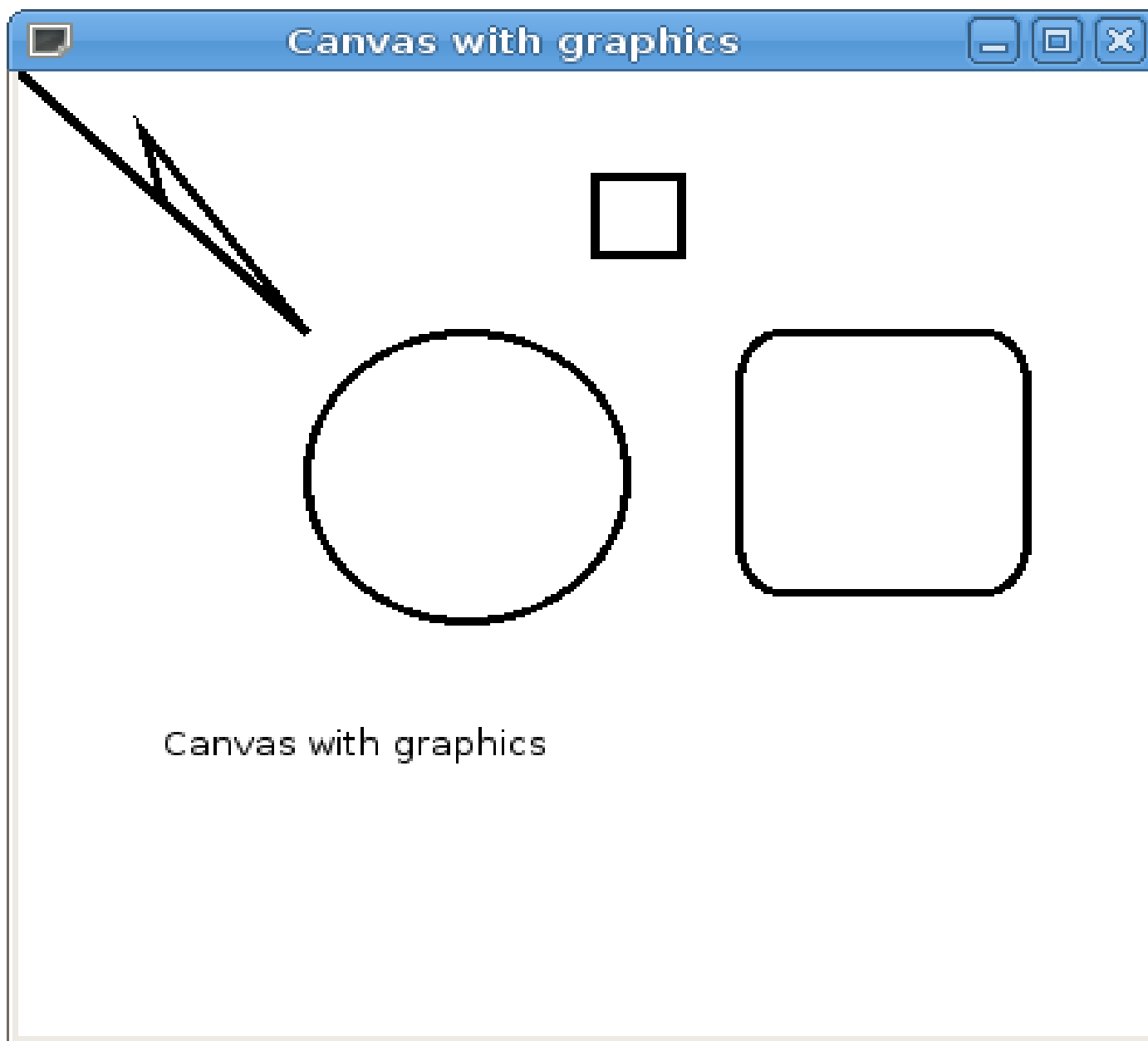
SWT Приложение с Canvas



GC разполага с методи за изчертаване на произволни обекти

```
GC gc = new GC(canvas);
gc.setLineWidth(3);
gc.drawLine(0, 0, 100, 100);
gc.drawOval(100, 100, 111, 111);
gc.drawPoint(100, 100);
gc.drawPolygon(new int[] { 50, 50, 100, 100, 43, 23 });
gc.drawRect(200, 40, 30, 30);
gc.drawRoundRectangle(250, 100, 100, 100, 30, 40);
gc.drawText("Canvas with graphics", 50, 250);
gc.dispose();
```

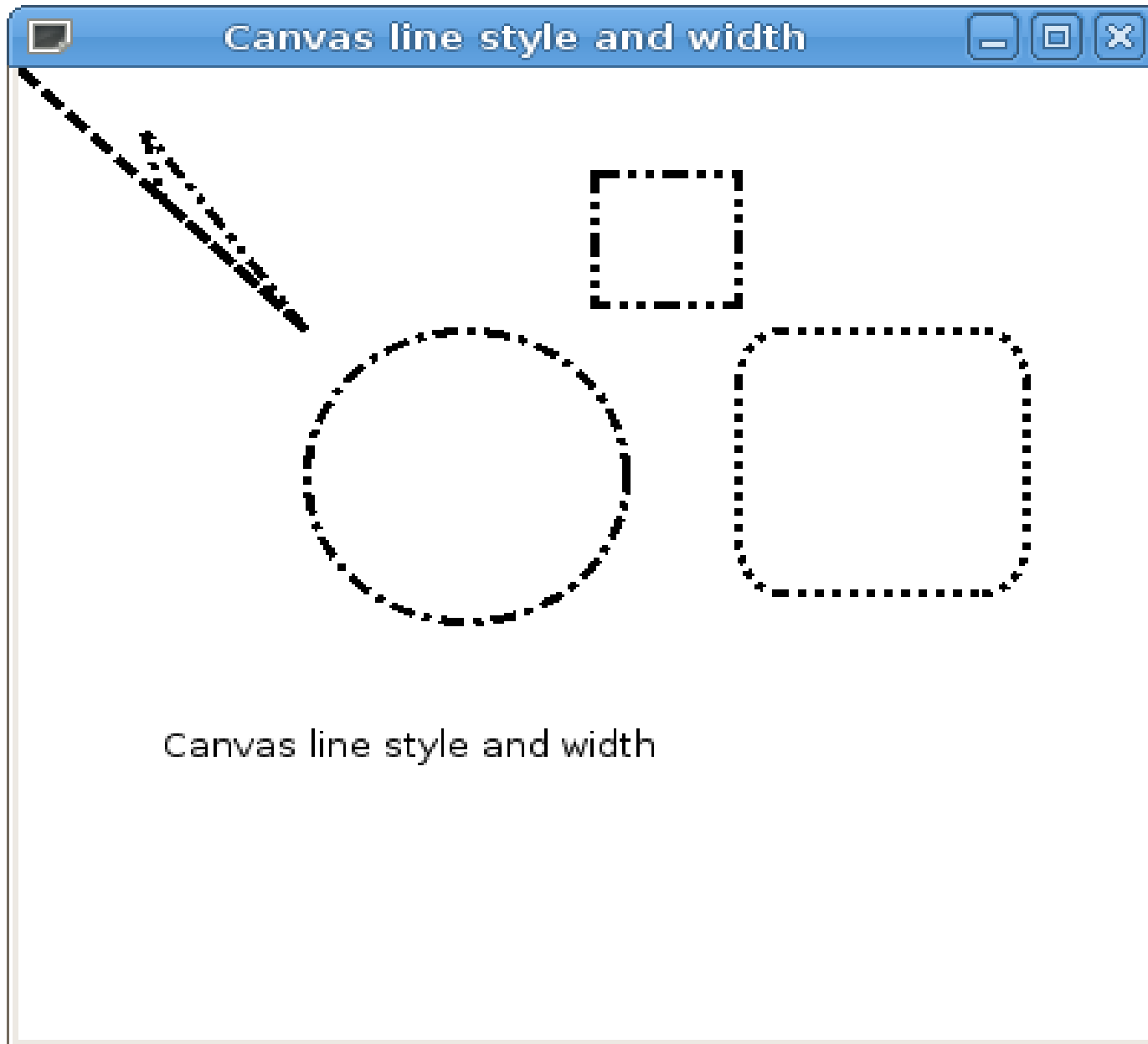

Canvas с различни графики



GC разполага с методи за изчертаване на произволни обекти

```
GC gc = new GC(canvas);
gc.setLineWidth(3);
gc.setLineStyle(SWT.LINE_DASH);
gc.drawLine(0, 0, 100, 100);
gc.setLineStyle(SWT.LINE_DASHDOT);
gc.drawOval(100, 100, 111, 111);
gc.setLineStyle(SWT.LINE_DASHDOTDOT);
gc.drawPolygon(new int[] { 50, 50, 100, 100, 43, 23 });
gc.drawRectangle(200, 40, 50, 50);
gc.setLineStyle(SWT.LINE_DOT);
gc.drawRoundRectangle(250, 100, 100, 100, 30, 40);
gc.drawText("Canvas with graphics", 50, 250);
gc.dispose();
```

Canvas с различни графики



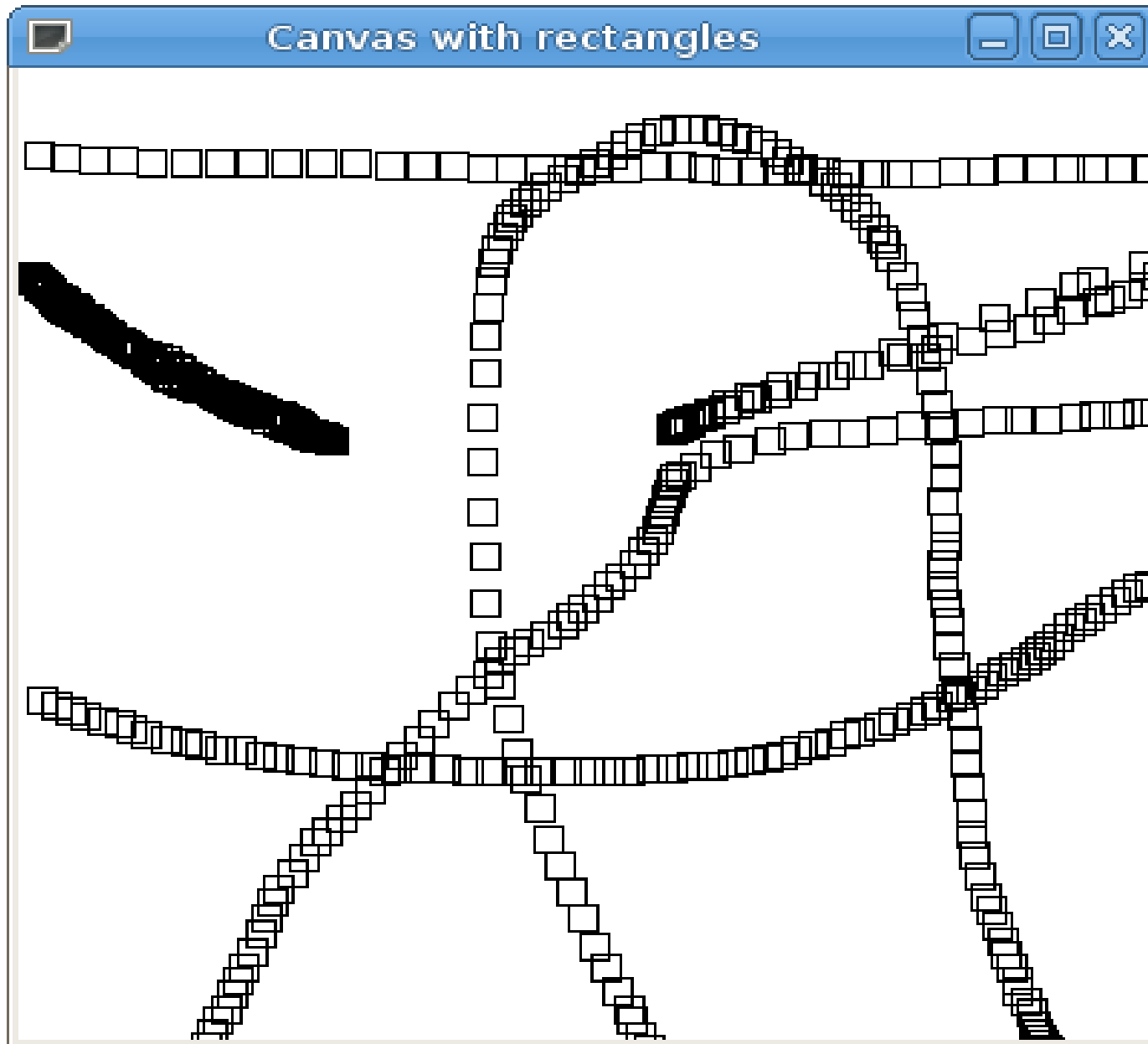
„Динамично“ изчертаване

Досегашните графики се изчертаваха “статично” - след стартиране на приложението. За да може размерът и формата на графиките да се контролира от потребителя се използват съобщенията от мишката.

```
Canvas canvas = new Canvas(shell, SWT.NONE);
shell.open();
final GC gc = new GC(canvas);
canvas.addMouseListener(new MouseMoveListener() {
    @Override
    public void mouseMove(MouseEvent e) {
        gc.drawRectangle(e.x, e.y, 10, 10);
    }
});
```

Програмата следи мишката при нейното движение и изчертава правоъгълник с размери 10 на 10 на мястото на мишката

Canvas при „динамично“ изчертаване

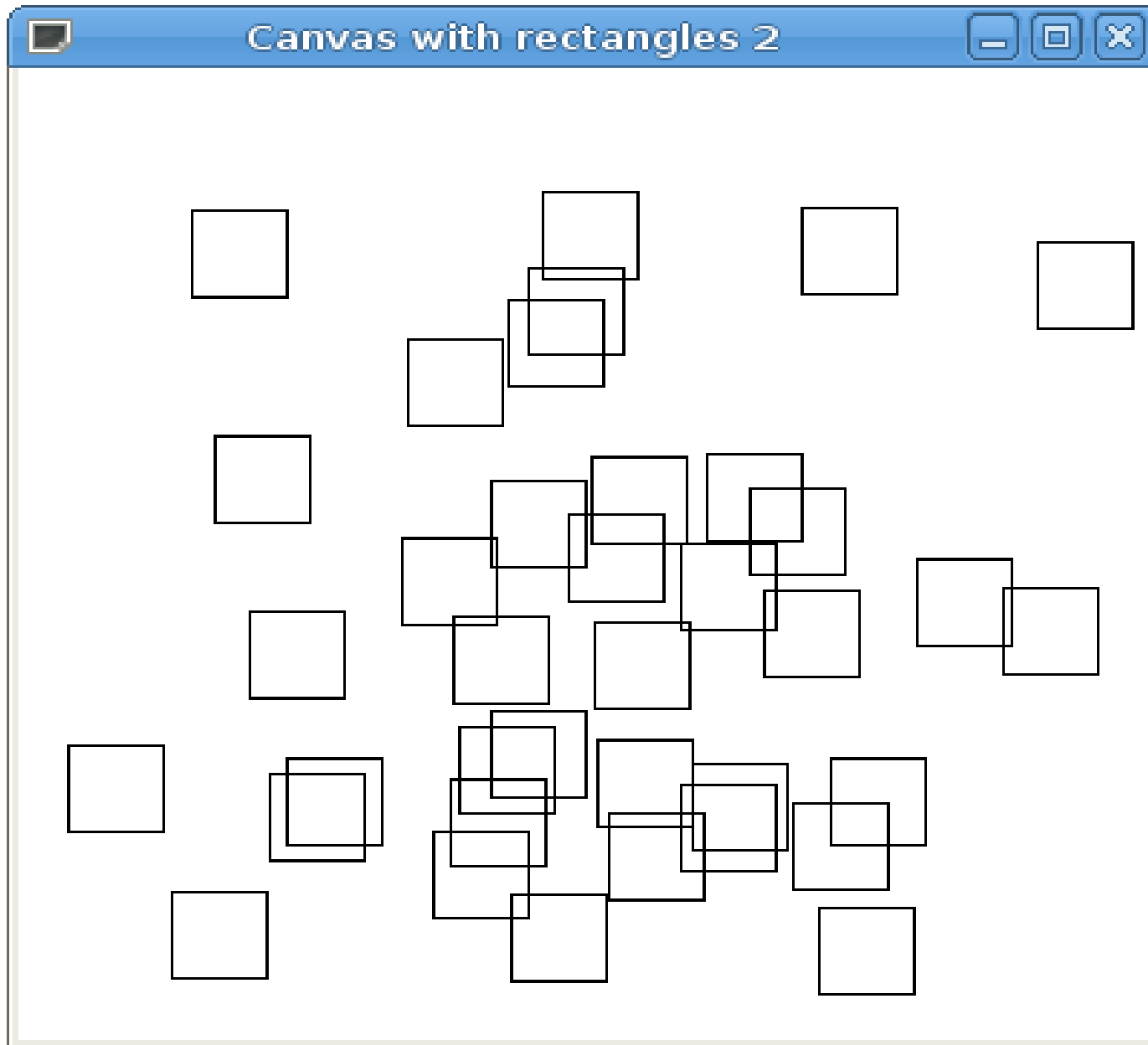


„Динамично“ изчертаване

Нито Canvas нито GC запаметяват в себе си какво обекти са изчертани на полето. Тази информация трябва да се съхранява от приложението, примерно в списък. В процеса на изчертаване на полето трябва да се пречертаят всички фигури наново.

```
final List<Point> points = new ArrayList<Point>();
canvas.addMouseListener(new MouseAdapter() {
    public void mouseDown(MouseEvent e) {
        points.add(new Point(e.x, e.y));
        canvas.redraw();
    }
});
canvas.addPaintListener(new PaintListener() {
    public void paintControl(PaintEvent e) {
        for(Point point: points) {
            e.gc.drawRectangle(point.x, point.y, 33, 33);
        }
    }
});
```

Canvas при „динамично“ изчертаване



„Динамично“ изчертаване

```
1  final List<Point> points = new ArrayList<Point>();
2  canvas.addMouseListener(new MouseAdapter() {
3      public void mouseDown(MouseEvent e) {
4          points.add(new Point(e.x, e.y));
5          canvas.redraw();
6      }
7  });
```

Ред 4-5: При натискане бутон на мишката, координатите на мишката при натискането се запамятват в списък – `points`. Цялото поле се прерисува защото вече имаме нова точка.

„Динамично“ изчертаване

```
1 canvas.addPaintListener(new PaintListener() {
2     public void paintControl(PaintEvent e) {
3         for(Point point: points) {
4             e.gc.drawRectangle(point.x, point.y, 33, 33);
5         }
6     }
7 });
```

Ред 3-5: Обхождат се всички запаметени точки и се пречертават правоъгълниците.

- SWT използва компоненти предоставени от операционната система . Ако дадена компонента не съществува в операционната система то тя се емулира.
- SWT предоставя изключително богат набор от компоненти. Пълният списък може да бъде намерен на:

<http://help.eclipse.org/help32/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/swt/widgets/package-summary.html>

<http://help.eclipse.org/help32/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/swt/custom/package-summary.html>

- Използват се стратегии за позициониране позволяващи динамично преразпределение на компонентите деца. Пълното описани може да бъде намерено на:

<http://help.eclipse.org/help32/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/swt/layout/package-summary.html>

Подходящ пример е:

<http://www.eclipse.org/swt/examples.php>

- SWT разчита на правилното управление на системните ресурси. Поради тази причина ако приложението създава ресурси то е задължено и да ги унищожи

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.