Curs 9

Introducere in applet-uri

Applet-ul este poate cea mai importanta dintre aplicațiile Java, si acest subiect nu poate fi cuprins intr-un capitol. Aşa că vom prezenta doar o privire de ansamblu asupra programării applet-uri.

Applet-urile folosesc o arhitectură unică, ce necesită folosirea unor tehnici de programare speciale. Una din aceste tehnici este tratarea evenimentelor. Evenimentele sunt modul in care un applet primeşte o intrare de la orice context exterior. Atât conceptul de applet cât şi cel de eveniment este foarte mare. În cele ce urmează vor fi tratate doar fundamentele acestor două concepte.

Applet-urile diferă de programele precedente deoarece ele sunt mici programe a căror scop este să ruleze pe Web. Iată un mic exemplu de acest tip de program:

```
import java.awt.*;
import java.applet.*;
public class OneApplet extends Applet
{
      public void init()
      {}

      public void paint(Graphics g)
      {
            g.drawString("First Applet.", 20, 20);
      }
}
```

Acest program începe cu două declarații de **import**. Prima importă clasele din pachetul AWT adică Abstract Window Toolkit. Applet-urile interacționează cu utilizatorul folosind aceste clase AWT, iar nu prim consolă. AWT conține o serie de clase ce permit lucrul cu ferestre şi o desenarea unei interfețe grafice.

Următoarea incluziune **import** se referă la pachetul **applet**. Acest pachet conține clasa *Applet*, iar orice applet creat trebuie să moştenească această clasă.

Clasa ce moşteneşte clasa *Applet*, este *OneApplet*. Această clasă ar trebui să fie declarată *public*, deoarece va fi accesată din afara ei.

Clasa conține o metodă *paint()*. Această metodă este definită de clasa *Component* din AWT, ce este o clasă părinte pentru Applet şi trebuie să fie suprascrisă de *OneApplet*. Metoda *paint()* este apelată ori de câte ori va avea loc reafişarea. Această reafişare poate surveni din mai multe motive. De exemplu, fereastra în care applet-ul rulează, este suprascrisa de alta fereastră, sau este mişcată. Sau poate fereastra este minimizată, şi apoi restaurată.

De asemenea *paint()* este apelată la pornirea applet-ului. Metoda *paint()* are un parametru de tip **Graphics**. Acest parametru conține contextul grafic în care rulează applet-ul.

În interiorul *paint()*, se află o metodă *drawString()* , care este un membru al clasei *Graphics*. Această metodă afişează un *String* la coordonatele X,Y. Forma ei generală este:

```
void drawString(String message, int x, int y)
```

Mesajul `message` va fi afişat pe fereastră la coordonatele *x*,*y*. Colțul stânga sus are coordonatele 0,0. De asemenea, de reținut că, applet-ul nu conține o metodă *main()*.

Spre deosebire de majoritatea programelor, execuția unui applet nu începe cu *main*. Pentru a putea porni un applet, acesta trebuie inclus într-un fișier HTML, pentru a rula în browser, sau se poate folosi un program auxiliar **appletviewer**.

Pentru a include un applet într-un browser, trebuie să modificăm un fișier HTML, adăugând un tag APPLET, astfel:

```
<html>
<applet code="OneApplet" width=200 height=60>
</applet>
</html>
```

Numele `OneApplet` vine de la fişierul *.class,* ce a fost obținut prin comanda:

```
javac OneApplet.java
```

Pe lângă acest nume, mai există `width=200 height=60` ce înseamnă dimensiunea suprafeței de afişare alocată acestui applet. Pentru a executa programul `OneApplet`, se poate apela următoarea comandă:

```
C:\>appletviewer OneApplet.html
```

Pentru a eficientiza rularea acestui applet, există o cale şi mai simplă: se include codul HTML în codul sursă, astfel:

```
import java.awt.*;
import java.applet.*;
/*
<applet code="OneApplet" width=200 height=60>
</applet>
*/
public class OneApplet extends Applet
{
      public void init()
      {}

      public void paint(Graphics g)
      {
            g.drawString("First Applet.", 20, 20);
      }
}
```

In felul acesta, codul HTML ce apare în comentariu la începutul programului, este folosit de **appletviewer** pentru a rula direct programul astfel:

```
C:\>appletviewer OneApplet.java
```

Astfel, în urma compilării şi rulării,

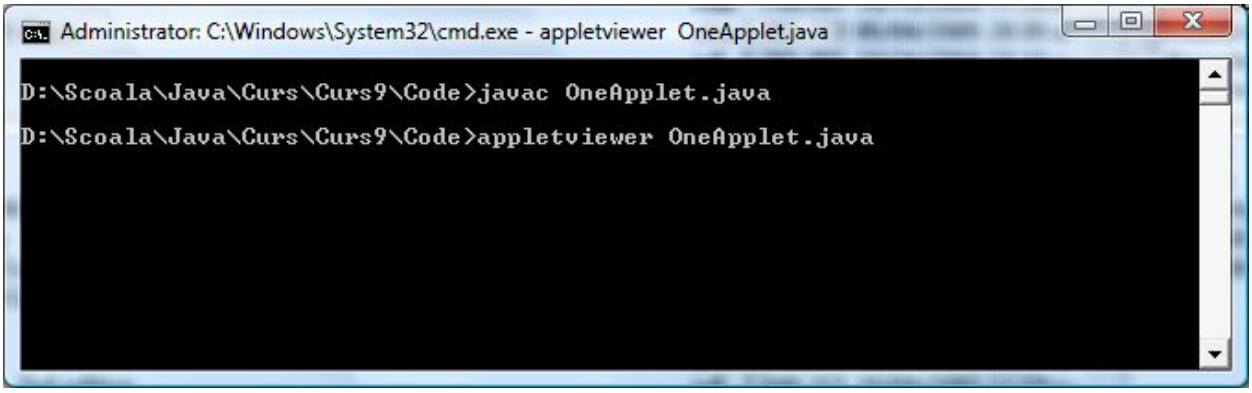

Figura 1. Paşii pentru rularea unui applet

obținem următoarea fereastra în care rulează applet-ul:

Figura 2. Interfața grafica a unui applet

După cum reiese şi din figura de mai jos, este un fel de *Panel*, el moştenind această clasă din pachetul java.awt. ca orice *Panel*, un *Applet*, poate conține componente de UI - user interface şi poate folosi toate modelele de desenare şi tratare a evenimentelor, specifice unei clase *Component*.

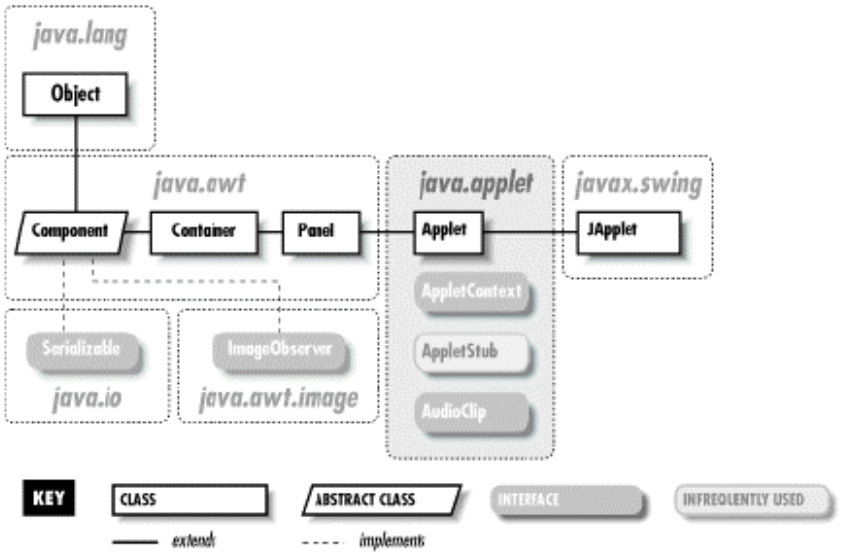

Figura 3. Pachetul java.applet

Pe lângă clasele AWT, există o librărie despre care vom vorbi în următorul capitol pe larg, denumită Swing. Aceasta este o alternativă la componentele AWT. De exemplu, pe lângă butoane, etichete, etc, Swing oferă paneluri pe care se poate efectua scroll, arbori, şi tabele. Practic este o extensie a acestor componente AWT.

Organizarea applet-urilor

Arhitectura unui applet

Un applet este un proces bazat pe ferestre. În astfel de arhitecturi, există câteva concepte, ce nu sunt prezente în programele ce rulează în consolă.

În primul rând, applet-urile sunt bazate pe evenimente şi seamănă cu un set de funcții de întrerupere. Un applet va aştepta până când un eveniment apare. Sistemul va notifica acest applet, apelând o metodă de tratare a evenimentului apărut. Odată ce aceasta are loc, applet-ul va efectua instrucțiunile din metodă, şi va preda înapoi controlul sistemului. În situațiile în care un applet trebuie să efectueze mai multe sarcini, până să returneze controlul sistemului, pentru a evita „înghețarea" ferestrei, putem lucra, sau efectua acele sarcini pe un alt fir de execuție.

În al doilea rând, utilizatorul este acela care interacționează cu applet-ul. În programe ce rulează în consolă, programul aşteaptă intrări, şi îl notifică pe utilizator. Aici, utilizatorul va interacționa cu applet-ul, după bunul lui plac. Aceste interacțiuni sunt transmise applet-ului sub forma unor evenimente. De exemplu, când utilizatorul apasă un buton al mouse-ului, se generează un eveniment de mouse. Dacă utilizatorul apasă o tastă când focus-ul este pe fereastra applet-ului, se va genera un eveniment de tastatură. Atunci când utilizatorul interacționează cu un buton, sau un check-box, sau orice alt control, se va genera un eveniment corespunzător.

Ce trebuie să conțină un applet

Deşi *OneApplet*, prezentat mai sus, este un applet real, el nu este complet, pentru că nu conține elementele necesare majorității applet-urilor. Metodele întâlnite în majoritatea acestor applet-uri sunt: *init()*, *start()*, *stop()* şi *distroy()*. Acestea sunt metodele definite de clasa *Applet*. A cincea metodă este *paint()*, şi este moştenită din clasa *Component*.

Mai jos avem un exemplu de implementare a unui *Applet*:

```
//Structura de baza a unui applet
import java.awt.*;
import java.applet.*;
/*
<applet code="AppletStructure" width=300 height=100>
</applet>
*/
public class AppletStructure extends Applet
{
    //prima metoda apelata.
```

```
    public void init()
    {
    // iniţializări
    }
    public void start()
    {
    // sart sau reiau execuţia
    }
    // apelată la oprirea appletului.
    public void stop()
    {
    // suspendă execuţia
    }
    //apelată la terminarea execuţiei
    //este ultima metodă apelată
    public void destroy()
    {
    //activităţi de oprire
    }
    // la redesenarea ferestrei
    public void paint(Graphics g)
    {
    //reafişarea conţinutului ferestrei
    }
}
```

Aceste metode au in dreptul lor un comentariu cu referire la scopul lor (ce ar trebui sa facă atunci când sunt apelate) şi in ce context sunt apelate.

Iniţializarea şi terminarea unui applet

Atunci când este lansat un applet, următoarele acţiuni au loc:

1. Apel *init()*
2. Apel *start()*
3. Apel *paint()*

Când un applet îşi încheie execuţia, următoarele acţiuni au loc:

1. Apel stop()
2. Apel *destroy()*

Prima metodă ce se apelează este *init()* şi în interiorul acesteia se iniţializează variabilele clasei şi alte activităţi de iniţializare. Metoda *start()* este apelată imediat după *init()* sau pentru a reporni un applet care a fost oprit. De aceea *start()* poate fi apelată de mai multe ori pe parcursul rulării unui applet. Metoda *paint()* este folosită la redesenarea ferestrei în care rulează applet-ul si a elementelor din fereastră. Metoda *stop()* permite suspendarea instrucţiunilor din applet, inclusiv a thread-urilor copii.

Metoda *destroy()* este apelată la încheierea totală a execuţiilor din applet.

Redesenarea

Ca regulă generală, un applet va scrie în fereastra **doar când** metoda *paint()* este apelată de sistem. Problema care se pune este următoarea: cum poate un applet să cauzeze aceste update-uri? De exemplu, dacă un applet afişează un banner în mişcare, ce mecanism permite acelui applet să updateze fereastra de fiecare dată? Nu se poate crea o buclă infinită în metoda *paint(),* pentru că ar bloca astfel sistemul. Metoda care permite redesenarea unor informații, sau actualizarea lor, este *repaint()*.

Metoda *repaint()* este definită în clasa *Component* din AWT. Aceasta face ca sistemul să execute un apel al funcției *update(),* metodă ce va apela *paint()*. Dacă o parte a applet-ului trebuie să afişeze un String, poate stoca acest mesaj într-o variabilă şi apela *repaint()*. Metoda *paint()* va prelua acea variabilă şi o va afişa folosind *drawString()*. Există două metode de redesenare:

```
void repaint()
void repaint(int left, int top, int width, int height)
```

În a doua metodă, coordonatele din colțul stânga sus al regiunii ce va fi redesenată, sunt date de primii doi parametrii. Dimensiunea regiunii este dată de ultimii doi parametrii. Aceste dimensiuni sunt specificare în pixeli.

Metoda *update()* este definită în Clasa Component, şi este apelată când applet-ul a cerut ca o porțiune din fereastră să fie redesenată. Există metode de a suprascrie această funcție, astfel încât funcționalitatea să fie mult mai mult decât un simplu apel ulterior al metodei *paint()*.

```
import java.awt.*;
import java.applet.*;
/*
<applet code="Banner" width=300 height=50>
</applet>
*/
//clasa implementeaza Runnable
//pentru ca vom lucra cu thread-uri
public class Banner extends Applet implements Runnable
{
      String msg = " Java permite afișarea unui banner.";
      Thread t;
      boolean stopFlag;
      // Inițializarea threadului cu null
      public void init()
      {
            t = null;
      }
      // start al appletului
     //folosit pentru a porni threadul t
      public void start()
      {
            t = new Thread(this);
```

```
            stopFlag = false;
            t.start();
        }
      //când threadul rulează
        public void run()
        {
            char ch;
            //Se afişează bannerul
            for( ; ; )
            {
                try
                {
                    repaint();
                    Thread.sleep(90);
                    ch = msg.charAt(0);
                    msg = msg.substring(1, msg.length());
                    msg += ch;
                    if(stopFlag)
                    break;
                }
            catch(InterruptedException exc)
            {}
            }
        }
        // pauza pe banner
        public void stop()
        {
            stopFlag = true;
            t = null;
        }
        // afişarea banner-ului
        public void paint(Graphics g)
        {
            g.drawString(msg, 50, 30);
        }
}
```

Clasa *Banner* extinde clasa *Applet*, dar implementează şi Runnable. Aceasta pentru că, applet-ul va crea un al doilea thread ce va fi folosit la actualizarea mesajului. Mesajul care va fi afişat, este cel din variabila *String msg*. Firul care se ocupă cu aceste modificări este dat de variabila *t*.

Variabila booleană *stopFlag*, este folosită pentru a opri applet-ul. În interiorul *init()* variabila este iniţializată cu null, urmând ca la pornirea applet-ului şi anume la apelul funcţiei *start()*, această variabilă sa fie *false* iar thread-ul să pornească.

Metoda care este cea mai interesantă este *run()*. Aici se va modifica variabila *msg*, astfel ca primul caracter din String să fie pus pe ultima poziţie. Aceste acţiuni au loc într-o buclă infinită, ceea ce ar trebui sa „înghețe" fereastra. Cum firul care se ocupa cu apelul metodei *paint()* este cel principal, funcţionarea nu va avea acest impediment. Când se încheie execuţia applet-ului, respectiv se apelează metoda *stop()*, acelaşi lucru se întâmplă şi cu thread-ul t, datorită variabilei `stopFlag`.

Cum se executa un applet

Un applet rulează în general într-un browser. Plugin-ul Java din browser controlează lansarea şi execuția applet-ului. Browser-ul are de obicei şi un interpretator JavaScript, care poate rula cod JavaScript în pagină.

Plugin-ul Java

Acest software, creează un thread pentru fiecare applet. Va lansa un applet într-o instanță de Java Runtime Enviroment (JRE). În mod normal, toate applet-urile rulează în aceeaşi instanță de JRE. Plugin-ul de Java, va porni o instanță de JRE în următoarele cazuri:

1. Când un applet cere să fie executat în versiunea corespunzătoare deJRE
2. Când un applet specifică proprii parametrii pentru JRE, de exemplu, mărimea heap-ului. Un nou applet foloseşte JRE existent, dacă specificațiile sunt o submulțime a specificațiilor JRE existent.

   Un applet va rula într-o instanță de JRE existentă, dacă toate condiițiile sunt îndeplinite:

1. Versiunea de JRE cerută de applet se potriveşte cu cea existentă.
2. Parametrii de start ai JRE, satisfac cerințele applet-ului.

Următoarea diagrama prezintă modul în care applet-urile sunt executate în JRE:

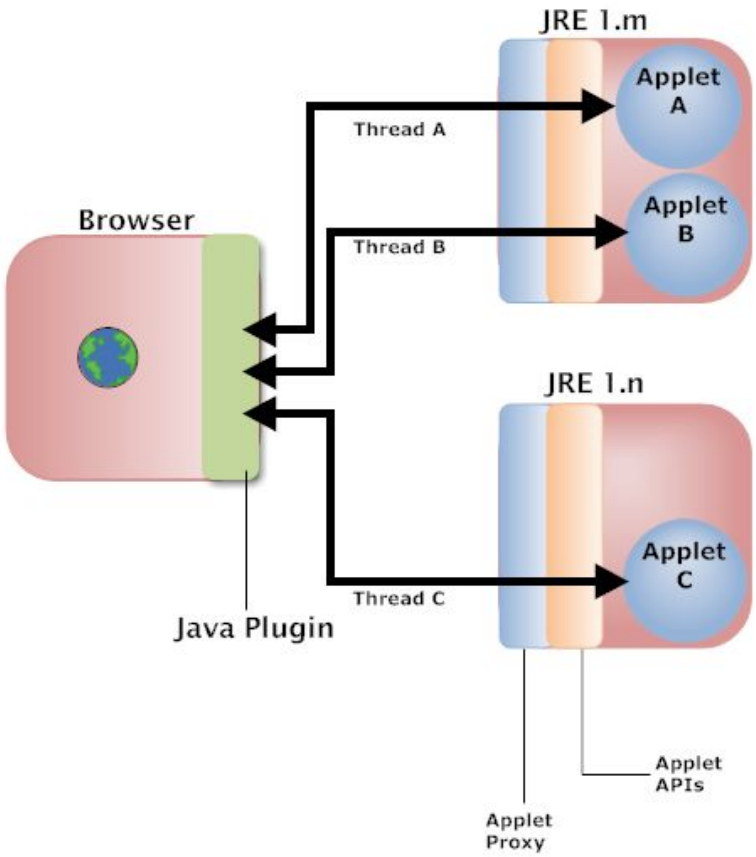

Figura 4. Execuția unui applet în JRE

Applet-urile pot invoca funcții prezente în pagina web. Funcțiile JavaScript pot de asemenea, invoca metode ale unui applet, încorporat în aceeaşi pagină web. Plugin-ul Java şi interpretatorul JavaScript vor orchestra aceste apeluri din cod Java în cod JavaScript şi viceversa.

Plugin-ul Java este multi-threaded, în timp ce interpretatorul JavaScript rulează pe un singur fir. Pentru a evita probleme legate de fire de execuție, mai ales când mai multe applet-uri rulează simultan, apelurile între cele două limbaje trebuie să fie scurte, şi să nu conțină, pe cât posibil bucle.

Folosirea status-ului din fereastră

Pe lângă afişarea informațiilor într-o fereastra, un applet poate afişa mesaje şi în bara de status a ferestrei în care applet-ul rulează. Pentru a realiza acest lucru, se apelează metoda showStatus(), care este definită de clasa Applet. Apelul aceste metode este:

```
void showStatus(String msg)
```

unde variabila *msg* reprezintă mesajul de afişat.

Fereastra de status, se poate folosi la informarea utilizatorului, despre acțiunile ce au loc în applet, de a sugera opțiuni, de a raporta erori, etc. Poate, fi folosita, de asemenea, la debug, pentru a afişa informații despre variabile, etc.

Mai jos este o simpla metoda ce afișează în bara de status un mesaj:

```
import java.awt.*;
import java.applet.*;
/*
<applet code="StatusWindow" width=300 height=50>
</applet>
*/
public class StatusWindow extends Applet
{
    //afişarea unui mesaj in status
    public void paint(Graphics g)
    {
        g.drawString("Acesta este un mesaj afişat in fereastra.", 10,
20);
        showStatus("Acesta este un mesaj afişat în status bar");
    }
}
```

Iată rezultatul rulării acestui program:

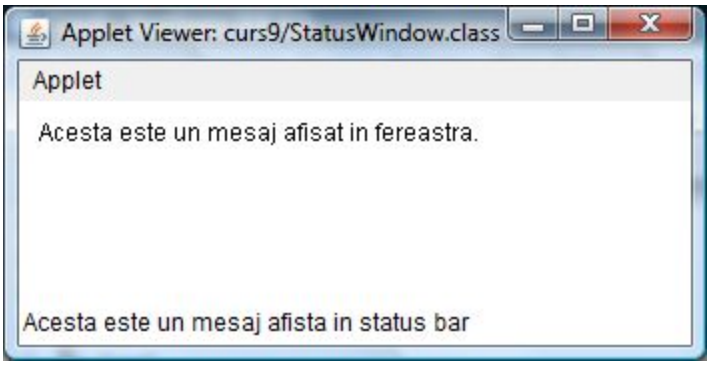

Figura 5. Mesaj in status bar

Transmiterea parametrilor către un applet

Pentru a transmite parametrii unui applet, se foloseşte atributul PARAM al tag-ului APPLET. Acest atribut va specifica numele parametrului şi valoarea sa. Pentru a afla valoarea unui parametru, se foloseşte *getParameter()* definită în clasa *Applet*. Apelul acestei metode este:

```
String getParameter(String paramName)
```

Aici, `paramName` este numele parametrului. Funcția va returna valoarea parametrului specificat, sub forma unui obiect *String*. Pentru valori booleane, va trebui să convertim reprezentările *String* într-un format intern. Dacă un parametru nu este găsit, valoarea *null* este returnată.

Iată un exemplu de folosire a parametrilor:

```
// Transmiterea parametrilor unui applet.
import java.awt.*;
import java.applet.*;
/*
<applet code="Param" width=300 height=80>
<param name=author value="John Smith"/>
<param name=purpose value="Demonstrate Parameters"/>
<param name=version value=43 />
</applet>
*/
public class Param extends Applet
{
      String author;
      String purpose;
      int ver;
      public void start()
      {
            String temp;
            author = getParameter("author");
```

```
            if(author == null) author = "not found";
            purpose = getParameter("purpose");
            if(purpose == null) purpose = "not found";
            temp = getParameter("version");
            try
            {
                  if(temp != null)
                        ver = Integer.parseInt(temp);
                  else
                        ver = 0;
            }
            catch(NumberFormatException exc)
            {
                  ver = -1; // error code
            }
      }
      public void paint(Graphics g)
      {
            g.drawString("Purpose: " + purpose, 10, 20);
            g.drawString("By: " + author, 10, 40);
            g.drawString("Version: " + ver, 10, 60);
      }
}
```

Locul în care sunt declarați parametrii ce vor fi transmiși applet-ului, la pornirea acestuia este în cadrul tag-ului <applet>:

```
<param name=author value="John Smith"/>
<param name=purpose value="Demonstrate Parameters"/>
<param name=version value=43 />
```

Odată ce parametrii au fost preluați la startul applet-ului, valorile lor vor fi folosite la afișarea pe ecran, in metoda *paint()*. Iată rezultatul:

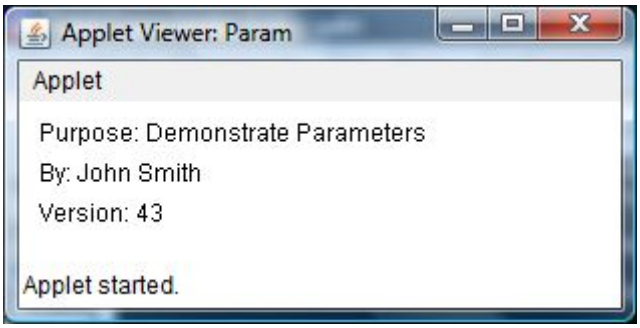

Figura 6. Parametrii unui applet

Clasa Applet

Toate applet-urile moştenesc clasa *Applet*. La rândul ei, clasa *Applet* moștenește următoarele clase definite în AWT: Component, Container şi Panel. În acest fel, un applet poate avea acces la funcționalitatea completă oferită de AWT.

Pe lângă aceste metode, *Applet* mai conține câteva funcții ce permit controlul total al applet-ului funcții ce sunt descrise mai jos:

Metoda	**Descriere**
void destroy()	Apelată de browser înainte de terminarea applet-ului. Applet-ul va suprascrie această metodă pentru a curăța alte obiecte.
AccessibleContext getAccessibleContext( )	Metodă ce returnează contextul de accesibilitate al obiectului ce invocă această metodă.
AppletContext getAppletContext( )	Metodă ce returnează contextul asociat unui applet.
String getAppletInfo()	Returnează un string ce descrie applet-ul
AudioClip getAudioClip(URL url)	Returnează un obiect de tip *AudioClip* ce încapsulează clipul audio găsit la locația specificată prin *url*.
AudioClip getAudioClip(URL url, String clipName)	Returnează un obiect de tip *AudioClip* ce încapsulează clipul audio găsit la locația specificată prin *url* şi având numele *clipName*.
URL getCodeBase( )	Returnează URL asociat cu applet-ul.
URL getDocumentBase( )	Returnează URL al documentului HTML ce invocă applet-ul.
Image getImage(URL url)	Returnează un obiect imagine ce încapsulează imaginea găsită, la o locație specificată prin *url*.
Image getImage(URL url, String imageName	Returnează un obiect imagine ce încapsulează imaginea găsită, la o locație specificată prin *url* şi având ca nume *imageName*.
Locale getLocale()	Returnează un obiect Locale ce este folosit în clase şi metode ce folosesc setările specifice unei regiuni.
String getParameter(String paramName)	Această metodă returnează un parametru asociat cu *paramName* sau null dacă nu este găsit acel parametru.
String[ ] [ ] getParameterInfo( )	Returnează o tabelă *String* ce descrie parametrii recunoscuți de către applet. Fiecare element din tabelă trebuie să fie format din trei string-uri ce conțin numele parametrului, descrierea tipului de dată, şi explicații cu privire la scopul lui.
void init()	această metodă este apelată la începutul execuției unui applet şi este prima metodă apelată.
boolean isActive()	Această metodă returnează *true* dacă applet-ul a pornit, sau false dacă a fost oprit.
static final AudioClip newAudioClip(URL *url*)	Această metodă returnează un *AudioClip* ce încapsulează clipul audio găsit la *url*-ul specificat. Această metodă este similară cu *getAudioClip*, însă este statică.

void play(URL url)	Dacă un audio clip este găsit la locația specificată de *url*, atunci clipul este rulat.
void play(URL, String clipName)	Dacă un audio clip este găsit la locația specificată de *url*, cu numele specificat de *clipName*, atunci clipul este rulat.
void resize(Dimension dim)	Redimensionează applet-ul conform dimensiunilor specificate de dim. *Dimension* este o clasă ce provine din java.awt. Conține două câmpuri: *width* şi *height*.
void resize(int width, int height)	Redimensionează applet-ul conform dimensiunilor width şi height.
final void setStub(AppletStub stubObj)	Creează obiectul stubObj, ce reprezintă un locțiitor de cod, pentru applet. Această metodă este folosită de sistemul run-time şi nu este apelată de applet-ul în sine. Un locțiitor sau *stub* este o bucată de cod, ce asigură o legătură dintre applet şi browser.
void showStatus(String str)	Afişează *str* în fereastra status a browser-ului sau a ferestrei de afişare a applet-ului.
void start()	apelat de browser atunci când un applet trebuie să fi pornit. Este apelat automat după metoda *init()*.
void stop()	apelat de browser atunci când un applet trebuie să îşi suspende activitatea. Odată oprit, un applet poate fi repornit prin *start()*.

Operațiuni grafice într-un applet

Clasa *Graphics* are o mulțime de funcții pentru desenare. Pentru fiecare figură geometrică, dreptunghi, arc, elipsa, poligon, există o metodă specifică ce desenează fie conturul, fie interiorul acelei figuri.
Iată mai jos un exemplu de desenare a unui dreptunghi umplut cu o culoare:

```
//colorarea unui dreptunghi
import java.awt.*;
import java.applet.*;
public class PaintDemo extends Applet
{
    int rectX = 20, rectY = 30;
    int rectWidth = 50, rectHeight = 50;
    public void init()
    {
        resize(getPreferredSize());
    }
    public void paint(Graphics g)
    {
```

```
            g.setColor(Color.red);
            g.fillRect(rectX, rectY, rectWidth, rectHeight);
        }
        public Dimension getPreferredSize( )
        {
         return new Dimension(100, 200);
        }
}
```

După cum se vede desenarea are loc în metoda *paint*, dar redimensionarea în *init*. Dacă am plasa redimensionarea în *paint()* ar fi practic imposibil redimensionarea ferestrei. Se pune problema, unde trebuie să desenăm, în *paint()* sau altundeva?

Când să desenăm?

Există mai multe opțiuni, pe care le putem alege când efectuăm desenarea unui obiect. Cea mai corectă metodă este însă de a desena în metoda *paint(),* deoarece nu se poate desena într-o fereastră până când aceasta nu este creată şi plasată pe ecran. În plus, se poate ca metodele să dureze mult, ceea ce blochează afişarea unor obiecte pe fereastra. Este recomandat ca în celelalte metode să se modifice parametrii obiectelor ce vor fi afişate, şi să se recheme *paint()* prin intermediul metodei de actualizare.

Utilizarea fonturilor

Pentru lucrul cu fonturi, în Java există două clase principale, *Font* şi *FontMetrics*. Prima serveşte la setarea şi lucrului concret cu anumite fonturi, redimensionarea lor, etc. A doua serveşte la preluarea măsurilor unui font şi a altor dimensiuni ale fontului.

Mai jos se află un exemplu pentru utilizarea acestor clase, însă este doar începutul, sau o primă privire aruncată asupra acestor aspecte, pentru că există foarte multe alte funcționalități care nu sunt abordate aici.

```
// demonstrarea modului corect de desenare
import java.awt.*;
import java.applet.*;
public class DrawStringDemo extends Applet
{
    String message = "Hello Java";
    //aici vom seta fontul folosit in applet
    public void init()
    {
        Font f = new Font("Calibri" , Font.BOLD|Font.ITALIC, 24);
        setFont(f);
    }
    //aici va avea loc desenarea
    public void paint(Graphics g)
    {
        //preluarea fontului curent si a dimensiunilor
        FontMetrics fm = getFontMetrics(getFont( ));
```

```
        //folosirea dimensiunilor fontului, si a mesajului
        //pentru a afla poziția la care începem sa desenam
        int textX = (getSize( ).width - fm.stringWidth(message))/2;
        if (textX<0) // If dacă string-ul este prea mare
        textX = 0;
        //acelaşi lucru si pentru înălțime
        int textY = (getSize().height - fm.getLeading( ))/2;
        if (textY<0)
        textY = 0;
        //aici se desenează textul pe fereastra
        g.drawString(message, textX, text);
    }
}
```

In cele ce urmează este exemplificat modul in care putem adăuga, efecte, cum ar fi umbra unui text. De asemenea, scopul este de a exemplifica modul de lucru cu parametrii, deoarece datele legate de font sunt preluate din parametrii transmişi applet-ului:

```
import java.applet.*;
import java.awt.*;
/**
<applet code="DropShadow" width=300 height=80>
<param name=label value="Eticheta"/>
<param name=fontname value="Arial"/>
<param name=fontsize value=43 />
</applet>
*/
class Main extends Frame
{
    public static void main(String args[])
    {
        //aici cream un nou applet
        DropShadow dp = new DropShadow();
        //pe care îl pornim
        dp.start();
    }
}
public class DropShadow extends Applet
{
    /*eticheta ce va apare in fereastra */
    protected String theLabel = null;
    /*Lățimea si înălțimea */
    protected int width, height;
    /*Numele fontului*/
    protected String fontName;
    /*variabila ce modifică fontul */
    protected Font theFont;
    /*mărimea fontului */
    protected int fontSize = 18;
```

```
/*deplasamentul umbrei*/
protected int theOffset = 3;
/*daca am primit toţi parametrii */
protected boolean inittedOK = false;
/*pentru a verifica dacă iniţializarea e ok*/
public void init( )
{
    theLabel = getParameter("label");
    if (theLabel == null)
    throw new IllegalArgumentException("LABEL is REQUIRED");
    //ne ocupam de font
    fontName = getParameter("fontname");
    if (fontName == null)
    throw new IllegalArgumentException("FONTNAME is REQUIRED");
    String s;
    if ((s = getParameter("fontsize")) != null)
    fontSize = Integer.parseInt(s);
    if (fontName != null || fontSize != 0)
    {
        theFont = new Font(fontName, Font.BOLD + Font.ITALIC,
        fontSize);
        System.out.println("Name " + fontName + ", font " +
        theFont);
    }
    if ((s = getParameter("offset")) != null)
        theOffset = Integer.parseInt(s);
    setBackground(Color.green);
    inittedOK = true;
   this.resize(getPreferredSize());
}
public Dimension getPreferredSize( )
{
 return new Dimension(200, 100);
}
/** Paint method showing drop shadow effect */
public void paint(Graphics g)
{
    if (!inittedOK)
    return;
    g.setFont(theFont);
    g.setColor(Color.black);
    g.drawString(theLabel, theOffset+30, theOffset+50);
    g.setColor(Color.white);
    g.drawString(theLabel, 30, 50);
}
//ce fel de parametri sunt necesari
public String[][] getParameterInfo( )
{
```

```
            String info[][] = {
            { "label", "string", "Text to display" },
            { "fontname", "name", "Font to display it in" },
            { "fontsize", "10-30?", "Size to display it at" },
            };
            return info;
        }
    }
```

Totodată, în exemplul de mai sus, se vedea un alt mod de a porni un applet, din cadrul unei metode main(). Se renunță astfel la browser, sau aplicația ajutătoare *appletviewer*:

```
            DropShadow dp = new DropShadow();
            //pe care îl pornim
            dp.start();
```

Pe lângă aceste funcționalități, există foarte multe alte componente de GUI pe care le vom aborda într-un capitol următor. În cele ce urmează vom aborda modul în care applet-urile tratează evenimentele ce pot apare.

Tratarea evenimentelor

Applet-urile sunt programe controlate de evenimente. De aceea, tratarea evenimentelor este inima, oricărui applet. Cele mai multe evenimente la care applet-ul va răspunde sunt generate de utilizator. Aceste evenimente sunt transmise applet-ului în diverse moduri. Există mai multe tipuri de evenimente. Cele mai des întâlnite sunt cele generate de mouse, tastatură şi diverse controale cum ar fi, un buton. Aceste clase care se ocupă de evenimente sunt conținute în pachetul java.awt.event.

Înainte de începerea discuției, trebuie spus că modul în care evenimentele sunt tratate de către un applet, să modificat de la versiunea 1.0 la versiunea 1.1. Primele metode de tratare a evenimentelor, sunt încă implementate, dar au devenit învechite. Ceea ce vom descrie în continuare, sunt metodele noi de tratare a evenimentelor.

Încă odată, trebuie spus că, ceea ce va fi prezentat este o mică parte din întregul subiect, însă oferă o idee despre modul în care se lucrează cu aceste concepte.

Modelul delegării

Acest mod de abordare nou, se bazează pe un model de a delega evenimentele. Acesta defineşte mecanismele standard de a genera şi procesa evenimente. Conceptul este următorul: o *sursă* generează un eveniment şi trimite o notificare unuia sau mai multor *ascultători*. În această schemă, ascultătorii aşteaptă pur şi simplu până când primesc un eveniment. Odată primit, ascultătorul procesează evenimentul şi apoi returnează controlul. Avantajul este că separă conceptele de interfața cu utilizatorul. O interfață cu utilizatorul poate *delega* procesarea unui eveniment unei alte părți de cod.

Pentru a putea primi o notificare, *ascultătorii* trebuie să se aboneze la un eveniment.

Evenimente

În modelul delegării, un eveniment este un obiect ce descrie o schimbare de context. Poate fi generat în urma interacțiunii unei persoane cu elementele grafice din interfață (apăsarea unui buton, a unei taste, selectarea unui obiect dintr-o lista, etc). De asemenea poate fi provocat, artificial, de către proces.

Sursele evenimentelor

O sursă a unui eveniment este un obiect ce generează acel eveniment. O sursă trebuie să înregistreze ascultătorii pentru a aceştia să primească notificări legate de un eveniment. Aceasta este metoda care va realiza înregistrarea, sau abonarea:

```
public void addTypeListener(TypeListener el)
```

Aici *Type* este numele evenimentului, şi *el* este referința la un ascultător. De exemplu metoda care înregistrează un ascultător la un eveniment de tastatură este **addKeyListenter()**. Metoda care înregistrează mişcarea unui mouse este **addMouseMotionListener()**. Atunci când un eveniment are loc, toți ascultătorii care s-au abonat la acel eveniment sunt notificați, primind o copie a obiectului eveniment.

O sursă va trebui să ofere şi o metodă ce permite anularea înregistrării de la un anumit eveniment. Forma generală a metodei care realizează acest lucru este:

```
public void removeTypeListener(TypeListener el)
```

Aici *Type* este numele evenimentului de la care se face deînregistrarea. De exemplu, dacă se doreşte ştergerea unui ascultător de la tastatură, se va apela metoda **removeKeyListener()**.

Metodele ce adaugă sau şterge un eveniment, sunt oferite de sursele ce generează aceste evenimente. De exemplu, clasa *Component*, oferă metode pentru adăugarea şi ştergerea ascultătorilor la tastatură şi mouse.

Ascultători de evenimente

Un *ascultător*, este un obiect ce este notificat atunci când un eveniment are loc. Acest ascultător trebuie să îndeplinească două condiții:

1. Să fie înregistrat cu una sau mai multe surse pentru a primi notificări despre anumite evenimente.
2. Să implementeze metode ce să proceseze aceste notificări.

Metodele care primesc şi procesează notificări sunt definite într-o mulțime de interfețe din pachetul **java.awt.event**. de exemplu interfața **MouseMotionListener** defineşte metode pentru a primi notificări atunci când mouse-ul este mutat sau tastat. Orice obiect poate primi şi procesa aceste evenimente.

Clase de evenimente

Aceste clase reprezintă mecanismul de bază de tratare a evenimentelor în Java. La vârful ierarhiei se află clasa *EventObject*, care se găseşte în pachetul **java.util**. Aceasta este superclasa pentru toate evenimentele. Clasa *AWTEvent*, definită în cadrul pachetului **java.awt** este o subclasă a lui *EventObject*. Este clasa părinte pentru toate evenimentele din AWT.

Pachetul **java.awt.event** defineşte câteva tipuri de evenimente ce sunt generate de diverse elemente de interfață. În tabelul de mai jos sunt enumerate aceste evenimente.

Clasa de eveniment	**Descriere**
ActionEvent	Generat când un buton este apăsat, se efectuează dublu click pe o listă, sau se selectează un meniu.
AdjustmentEvent	Generat când un scroll bar este manipulat
Component Event	Generat când o componentă este ascunsă, mişcată sau redimensionată.
ContainerEvent	Generat când o componentă este adăugată sau ştearsă dintr-un obiect Container.
FocusEvent	Generat când o componentă primeşte sau pierde focusul tastaturii
InputEvent	O clasa abstractă părinte tuturor componentelor de evenimente de intrare
ItemEvent	Generat când se efectuează click pe un checkbox sau o listă, sau este Realizată o selecție de opțiune într-un meniu.
KeyEvent	Generat când intrarea este recepționată de la tastatură
MouseEvent	Generat când un mouse este trasat, sau mutat apăsat sau eliberat
TextEvent	Generat când valoarea unui câmp text se modifică.
WindowEvent	Generat când o fereastră este activată, închisă, deactivată, etc.

Pentru toate sursele mai sus menționate există p serie de ascultători, şi anume interfețe ce folosesc la captarea notificărilor evenimentelor produse.

Acestea sunt : ActionListener, AdjustmentListener, …, WindowListener.

Folosirea modelului de delegare

Pentru a programa acest model trebuie să respectăm următorii paşi:

1. Se implementează o interfață corespunzătoare în ascultător aşa încât va recepționa evenimentele corespunzătoare.
2. Implementarea codului astfel încât să înregistreze şi de-înregistreze (după caz) un ascultător, la şi de la anumite evenimente.

Orice sursă poate genera mai multe tipuri de evenimente. Fiecare eveniment trebuie înregistrat separat. De asemenea, un obiect poate fi înregistrat pentru a recepționa notificări de la mai multe evenimente. Pentru aceasta trebuie să implementeze toate interfețele cerute.

În continuare vom analiza tratarea evenimentelor mouse-ului.

Tratarea evenimentelor de mouse

Pentru a trata evenimentele de mouse, trebuie să implementăm interfețele **MouseListener** şi **MouseMotionListener.** Interfața MouseListener defineşte cinci metode. Dacă un buton al mouse-ului este apăsat, atunci este invocată metoda *mouseClicked()*. Atunci când mouse-ul se află în cadrul unei componente, cum ar fi o fereastră, este apelată metoda *mouseEntered()*, iar când părăseşte componenta, se apelează metoda *mouseExited()*. De asemenea, metodele *mousePressed()* şi *mouseReleased()* sunt apelate când un buton al mouse-ului este apăsat respectiv eliberat.
Iată forma metodelor:

```
void mouseClicked(MouseEvent me)
void mouseEntered(MouseEvent me)
void mouseExited(MouseEvent me)
void mousePressed(MouseEvent me)
void mouseReleased(MouseEvent me)
```

Interfața *MouseMotionListener()* defineşte două metode. Prima, *mouseDragged()* este apelată de mai multe ori când mouse-ul este trasat. Metoda *mouseMoved()* este invocată ori de cîte ori mişcăm mouse-ul. Formele acestor metode sunt:

```
void mouseDragged(MouseEvent me)
void mouseMoved(MouseEvent me)
```

Evenimentul MouseMoved transmite un parametru *me* ce descrie evenimentul. Cele mai folositoare metode din cadrul acestei clase sunt *getX()* şi *getY()* ce preiau coordonatele mouse-ului:

```
int getX()
intgetY()
```

Mai jos se află un eveniment, ce demonstrează folosirea evenimentelor de mouse. Acest applet va afişa coordonatele actuale ale mouse-ului în cadrul ferestrei. De fiecare dată când un buton este apăsat, este afişat cuvântul „Down”, iar când este eliberat, se afişează cuvântul „Up”. Dacă un buton este apăsat se va afişa în colțul stânga sus mesajul „Mouse clicked”.

Atunci când mouse-ul se află deasupra ferestrei, mesajul de notificare apare în colțul stânga sus. Atunci când se trasează mouse-ul, se afişează un mesaj „*” în coordonatele actuale ale mouse-ului. În toate aceste modificări, coordonatele mouse-ului vor fi salvate în *mouseX* şi *mouseY*. Aceste Variabile vor fi utilizate în metoda *paint()* pentru a afişa diversele mesaje.

```
import java.awt.event.*;
import java.applet.*;
import java.awt.*;
/*
<applet code="MouseWatcher" width=300 height=100>
</applet>
*/
public class MouseWatcher extends Applet
```

```
implements MouseListener, MouseMotionListener
{
   String msg = "";
   int mouseX = 0, mouseY = 0; //coordonatele
    Image image;
   public void init()
   {
        //la iniţializare
        //ne abonam pentru a fi notificaţi
        //la evenimentele mouse-ului
         addMouseListener(this);
         addMouseMotionListener(this);
   }
   // Tratare eveniment click de mouse
   public void mouseClicked(MouseEvent me)
   {
         mouseX = 0;
         mouseY = 10;
         msg = "Mouse clicked.";
         repaint();
   }
   // Tratare eveniment de intrare mouse
   public void mouseEntered(MouseEvent me)
   {
         mouseX = 0;
         mouseY = 10;
         msg = "Mouse entered.";
         repaint();
   }
   // Tratare eveniment de ieşire muse
   public void mouseExited(MouseEvent me)
   {
         mouseX = 0;
         mouseY = 10;
         msg = "Mouse exited.";
         repaint();
   }
   // Tratare eveniment de apăsare mouse
   public void mousePressed(MouseEvent me)
   {
         // salvare coordonate
         mouseX = me.getX();
         mouseY = me.getY();
         msg = "Down";

         repaint();
   }
   // Tratare eveniment de eliberare mouse
```

```
    public void mouseReleased(MouseEvent me)
    {
          // salvare coordonate
          mouseX = me.getX();
          mouseY = me.getY();
          msg = "Up";
          repaint();
    }
    // Tratare eveniment de trasare mouse
    public void mouseDragged(MouseEvent me)
    {
          //salvare coordonate
          mouseX = me.getX();
          mouseY = me.getY();
          msg = "*";
          showStatus("Dragging mouse at " + mouseX + ", " + mouseY);
          repaint();
    }
    // Tratare eveniment de mişcare mouse
    public void mouseMoved(MouseEvent me)
    {
          //afişarea pe status
          showStatus("Moving mouse at " + me.getX() + ", " +
          me.getY());
    }
    // se afişează mesajul la coordonatele salvate
    public void paint(Graphics g)
    {
         g.drawString(msg, mouseX, mouseY);

    }
}
```

In figura de mai jos se poate vedea rezultatul acțiunilor, notificate către handlerele de mouse, ce modifică variabila de mesaj şi coordonatele.

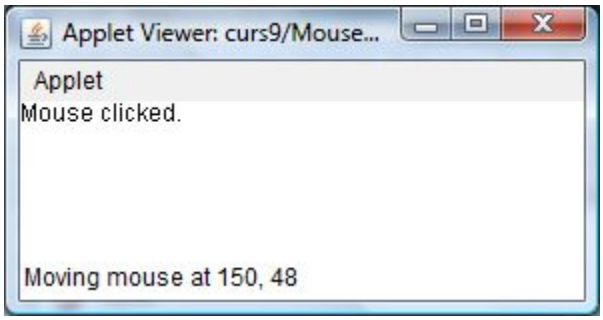

Figura 7. Tratarea evenimentelor unui mouse

În cadrul procedurii de *init()*, applet-ul se înregistrează ca ascultător la evenimentele mouse-ului. Aceasta se face prin cele două apeluri de *addMouseListener* şi *addMouseMotionListener*.

Ultimul exemplu se referă la tratarea evenimentelor tastaturii, şi este asemănător celui anterior. Diferența este că, abonarea se va face la evenimentele de tastatură, şi evident nu vom avea coordonate, de această dată. Totuşi, cele două programe pot fi combinate astfel ca ambele tipuri de evenimente să fie tratate.

```
import java.awt.event.*;
import java.applet.*;
import java.awt.*;
/*
<applet code="KeyWatcher" width=300 height=100>
</applet>
*/
public class KeyWatcher extends Applet
        implements KeyListener
{
   String msg = "";
    Image image;
   public void init()
   {
        //la initializare
        //ne abonam pentru a fi notificati
        //la evenimentele tastaturii
         addKeyListener(this);
        resize(800,100);
   }
    /* tratare evetiment tastare*/
     public void keyTyped(KeyEvent e) {
        msg = "KEY TYPED: " + e.paramString();
        repaint();
    }
     /* tratare eveniment apasare tasta */
    public void keyPressed(KeyEvent e) {
       msg =  "KEY PRESSED: " + e.paramString();
       repaint();
    }

    /*tratare eveniment eliberare tasta*/
    public void keyReleased(KeyEvent e) {
       msg = "KEY RELEASED: " + e.paramString();
       repaint();
    }

   // se afişează mesajul conform tastei apăsate
   public void paint(Graphics g)
   {
        g.drawString(msg,20,20);
   }
}
```