עיצוב ממשק משתמש גרפי
Graphic user interface (GUI)

מבוא
עיצוב ממשק משתמש גרפי (ממשק חלונאי) הוא חלק הכרחי ברבות מהאפליקציות המודרניות. קיים ויכוח ארוך שנים לגבי הנחיצות של ממשקים גרפיים לעומת ממשקים טקסטואליים פשוטים יותר. בדרך כלל לממשקים הגרפים, אם הם מעוצבים נכון, יש יתרון של עקומת-לימוד-שימוש תלולה יותר ואילו לממשקים הטקסטואליים יש יתרון במהירות שימוש ובעושר הביטוי. אפליקציות רבות משלבות בין שני הממשקים ומאפשרות למשתמש ליהנות מהיתרונות של שני העולמות.

הספרייה הסטנדרטית של Java כוללת מספר חבילות (packages) לתכנות ממשק חלונאי. החבילה המקורית נקראת AWT (עבור Abstract windows toolkit): חבילה זו מספקת שירותים בסיסיים יחסית. AWT הוצגה לראשונה בגרסה Java-1.0 אבל שופרה מאוד בגרסה Java1.1. בגרסה Java-1.2 נוספה חבילה חדשה בשם swing המספקת חלופות להרבה מהכלים של AWT אבל לא לכולם. ההמלצה כיום היא להשתמש בכלים החדשים של swing ובשירותים של AWT שאינם ניתנים ע"י swing. אנו נעקוב אחרי המלצה זו.

ישנן שתי סיבות עיקריות מדוע כדאי ללמוד להשתמש בכלי הממשק הגרפי של Java. הסיבה הראשונה כבר הוזכרה: מימוש ממשק חלונאי הוא ידע נחוץ היום. הסיבה השניה היא שהחבילות הגרפיות מהוות דוגמה טובה לעיצוב מונחה עצמים לא טריוויאלי. החבילות כוללות היררכית ירושה ענפה, רמות אבסטרקציה רבות ושימוש אינטנסיבי בפולימורפיזם.

אפליקציות חלונאיות ואפלטים

אפליקציה בעלת ממשק משתמש גרפי, היא תוכנית שכאשר מריצים אותה, נפתח חלון (או מספר חלונות) והאפליקציה מגיבה למאורעות (Event) שונים. מאורע יכול להיות לחיצה על כפתור, הזזה של עכבר, שינוי גודל של חלון וכ'. כתיבה של אפליקציה חלונאית דורשת כתיבת קוד המשודך למאורעות שונים בניגוד לאפליקציה רגילה, היא איננה מורכבת מקבוצת הוראות המופעלות לפי סדר ידוע מראש.

אפלט (Applet) הוא מושג שהומצא ע"י המתכנים של Java והוא מתאר תוכנית המורצת מתוך דפדפן רשת (Browser). אפלטים משובצים בתוך דפי HTML, וכאשר גולש מגיע אליהם יורד ה - bytecode של האפלט למחשבו של הגולש ומורץ ע"י מכונה ווירטואלית בתוך הדפדפן. בניגוד לשירותי רשת רבים המורצים על מחשבי השרת, האפלט מורץ ע"י מחשב הלקוח. לאפלט יש אפשרות לממשק משתמש דומה לזה של אפליקציה חלונאית אך עם מספר הגבלות.
היתרונות של אפלט לעומת אפליקציה רגילה הם חוסר הצורך בהתקנה, בטיחות ועדכון שקוף למשתמש. החיסרון הוא הזמן שלוקח ל - bytecode לרדת מהרשת כל פעם מחדש.

אפליקציה חלונאית פשוטה
התוכנית הבאה פותחת חלון פשוט עם כותרת

```
import javax.swing.*;

class Check {
   public static void main(String[] args) {
      JFrame frame = new JFrame("Check");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);//DO_NOTHING_ON_CLOSE
      frame.setSize(200,80);
      frame.setVisible(true);
   }
}
```

Jframe היא מחלקה המייצגת חלון עם מסגרת וכותרת. הפונקציה setDefaultCloseOperation מאפשרת לקבוע אם לחיצה על ה X בפינה הימינית עליונה, תסגור את החלון או לא. הפונקציה

setSize() קובעת את גודלו של החלון והפונקציה setVisible קובעת אם הוא גלוי או לא (בברירת המחדל היא מוסתר). "פלט" התוכנית יהיה:

כפי שניתן לראות, בניגוד לתוכנית רגילה, פונקצית ה - main לא הכילה סידרת הוראות לביצוע אלא הנחיות לסידור עצמים גרפים המגיבים למאורעות. במקרה זה קיים רק מאורע יחיד של סגירת חלון. אפשר לחשוב על אפליקציה חלונאית כתוכנית הנכנסת ללולאה אין סופית בה היא ממתינה למאורעות ומטפלת בהם. ה"לולאה" הזאת איננה גלויה למשתמש והיא מתחילה לאחר שכל ההוראות ב - main בוצעו.

דוגמה לapplet פשוט

כזכור, אפלט הוא תוכנית המורצת ע"י דפדפן. האפלט הפשוט ביותר נראה כך:

file: Check.java

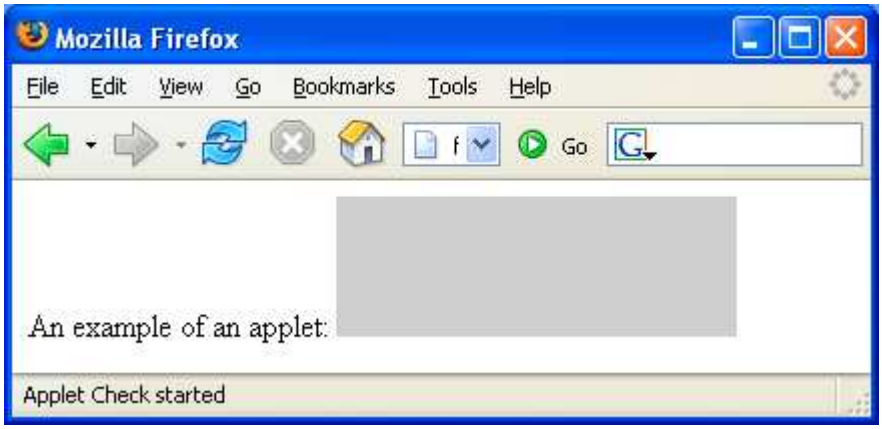
```
import javax.swing.*;
public class Check extends Japplet {}
```

אפלט נמצא תמיד בתוך דף HTML. הנה קוד HTML לדוגמה:

file: Stam.html

```
<html>
An example of an applet:
<applet code=Check width=50 height=70> </applet>
</html>
```

התג applet מציין את קובץ ה - class. אליו צריך להתייחס ואת גודל התצוגה של האפלט.
פלט:



אפלט שהוא גם אפליקציה

מבחינת ממשק משתמש, יש דמיון רב בין כתיבת קוד לאפליקציה חלונאית ואפלט. למעשה, ניתן לכתוב קוד אותו ניתן להריץ גם כאפלט וגם כאפליקציה. לדוגמה:

```
import javax.swing.*;
public class Check extends Japplet {
   //…
   public static void main(String[] args) {
      Japplet applet = new Check();
      Jframe frame = new Jframe("Check");
      frame.setDefaultCloseOperation(Jframe.EXIT_ON_CLOSE);
      frame.getContentPane().add(applet);
      frame.setSize(50, 70);
      applet.init();
      applet.start();
      frame.setVisible(true);
   }
}
```

לאחר קומפילציה, ניתן להריץ קוד זה כאפליקציה ע"י

```
$ java Check
```

או לראות אותו ע"י דפדפן כאפלט בתוך קובץ הHTML שכתבנו קודם.

אפליקציה עם שני כפתורים

התוכנית הבאה מדגימה יצירת חלון המכיל שני כפתורים:

```
import javax.swing.*;
import java.awt.*; // for FlowLayout

class Check {
   public static void main(String[] args) {
      JFrame frame = new JFrame("Check");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.getContentPane().setLayout(new FlowLayout());
      frame.getContentPane().add(new JButton("One"));
      frame.getContentPane().add(new JButton("Two"));

      frame.setSize(200,100);
      frame.setVisible(true);
   }
}
```

פלט:

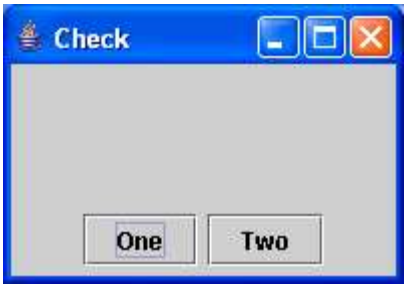
הפונקציה getContentPane() מאפשרת לקבל את ה"זגוגית" של החלון - מבנה הנתונים המחזיק את כל העצמים המונחים על החלון. הדבר הראשון שהוספנו לזגוגית הוא FlowLayout - מנהל סידור עצמים המסדר את העצמים משמאל לימין. לאחר מכן הוספנו שני כפתורים בעלי שמות מוגדרים.

אפליקציה המכילה שטח לציור

האפליקציה הבאה מדגימה כיצד ניתן לסדר עצמים שונים על החלון תוך שימוש בפנלים:

```
import javax.swing.*;
import java.awt.*;
class Check {
   public static void main(String[] args) {
      JFrame frame = new JFrame("Check");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container pane = frame.getContentPane();
      pane.setLayout(new BorderLayout());

      JPanel canvas = new JPanel(), panel = new JPanel();

       panel.add(new JButton("One"));
       panel.add(new JButton("Two"));

       pane.add(canvas,BorderLayout.CENTER);
       pane.add(panel,BorderLayout.SOUTH);

      frame.setSize(200,140);
      frame.setVisible(true);
   }
}
```

פלט:



בניגוד לאפליקציה הקודמת, הפעם בקשנו את זגוגית החלון רק פעם אחת והתייחסנו אליה כ - Container לאורך כל התוכנית. הפעם השתמשנו במנהל סידור אחר בשם BorderLayout. מנהל זה מאפשר למקם עצמים בכווני המפות - צפון, דרום, מזרח, מערב וגם במרכז. JPanel הוא אובייקט גרפי המאפשר לשים עליו אובייקטים גרפיים אחרים. ייצרנו שני אובייקטים גרפיים, אחד שימש כ - canvas - משטח ציור והשני בכדי למקם עליו את שני הכפתורים.

בשלב הבא מקמנו את שני הכפתורים על הפנל המיועד לכך. לאחר מכן מקמנו את פנל ה - canvas במרכז ואת פנל הכפתורים בדרום (בתחתית) החלון.

אפליקציה חלונאית המגיבה למאורעות

כפי שכבר נאמר, תכנות של אפליקציה חלונאית דורש שידוך של פעולות למאורעות. מאורע אחד יכול להיות משודך למספר פעולות, ופעולה אחת יכולה להיות משודכת למספר מאורעות.

בספריה הגרפית של Java המושג המרכזי המאפשר את השידוך של המאורעות לפעולות הוא המאזין - Listener. לכל אובייקט בו יכל להתרחש מאורע יש מבנה נתונים של מאזינים למאורע.

המאזינים הם אובייקטים המכילים פונקציה בעלת שם ידוע מראש. כאשר מתרחש המאורע, מודיע האובייקט לכל המאזינים הרשומים אצלו (נמצאים במבנה הנתונים שלו) שהמאורע התרחש ואז מופעלת הפונקציה שלהם.

הקוד הבא מדגים הוספת מאזין לאחד מהכפתורים של הדוגמה הקודמת. הפעולה שנגרמת ע"י לחיצה על הכפתור, היא פשוט הדפסת פרטי המאורע - שם הכפתור שנלחץ וזמן הלחיצה:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*; // for ActionListener & ActionEvent

class Check {
   public static void main(String[] args) {
      JFrame frame = new JFrame("Check");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         Container pane = frame.getContentPane();
         pane.setLayout(new BorderLayout());

         JPanel canvas = new JPanel(), panel = new JPanel();
         JButton b1 = new JButton("One");
         b1.addActionListener(new But1ActionListener());

         panel.add(b1);
         panel.add(new JButton("Two"));

         pane.add(canvas,BorderLayout.CENTER);
         pane.add(panel,BorderLayout.SOUTH);

      frame.setSize(200,140);
      frame.setVisible(true);
   }
}

class But1ActionListener implements ActionListener {
   public void actionPerformed(ActionEvent e) {
         System.out.println("command: "+ e.getActionCommand()
                              + "\ntime: "+ e.getWhen()+ "\nparams: "
                              + e.paramString());
   }
}
```

הפונקציה addActionListener של Jbutton מאפשרת להוסיף מאזין ללחיצה על הכפתור. המאזין צריך להיות אובייקט הממש את הממשק ActionListener וככזה לממש את הפונקציה actionPerformed. כאשר הכפתור נלחץ, הפונקציה actionPerformed של המאזין מופעלת. הפרמטר שנשלח לפונקציה הוא אובייקט מסוג ActionEvent המכיל את פרטי המאורע שהתרחש.

שימוש בתחביר של משתנה אנונימי

במבט חוזר על הדוגמה הקודמת, ניתן לראות שאפשר לשפר את הקוד. האבחנה הראשונה היא שהמחלקה But1ActionListener נוצרה רק לשימוש Check ולכן אמורה להיות inner class שלה. האבחנה השניה היא שהמחלקה נועדה לשימוש במקום יחיד בקוד, אין צורך לתת לה שם ואפשר להשתמש בתחביר של משתנה אנונימי:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*; // for ActionListener & ActionEvent

class Check {
   public static void main(String[] args) {
      JFrame frame = new JFrame("Check");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         Container pane = frame.getContentPane();
         pane.setLayout(new BorderLayout());

         JPanel canvas = new JPanel(), panel = new JPanel();
         JButton b1 = new JButton("One");
         b1.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {
                     System.out.println("command: "+ e.getActionCommand()
                                    + "\ntime: "+ e.getWhen()+ "\nparams: "
                                    + e.paramString());
                  }
            });

         panel.add(b1);
         panel.add(new JButton("Two"));

         pane.add(canvas,BorderLayout.CENTER);
         pane.add(panel,BorderLayout.SOUTH);

      frame.setSize(200,140);
      frame.setVisible(true);
   }
}
```

מספר פעולות המקושרות לאותו מאורע

כפי שכבר נאמר, מאורע אחד יכל להיות מקושר למספר פעולות. ניתן להוסיף לקוד הקודם עוד פקודה המוסיפה לכפתור מאזין נוסף המבצע פעולה אחרת:

```
class Check {
   public static void main(String[] args) {
         …
         b1.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {
                     System.out.println("command: "+ e.getActionCommand()
                                    + "\ntime: "+ e.getWhen()+ "\nparams: "
                                    + e.paramString());
                  }
            });
         b1.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {
```

```
                System.out.println("bla bla bla");
            }
        });
    …
  }
}
```

שימוש בעכבר

התוכנית הבאה מוסיפה מאזין לעכבר - בכל לחיצה על העכבר בתחום החלון, יודפסו הקואורדינטות שלו:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*; // for MouseListener & MouseEvent
class Check {
   public static void main(String[] args) {
      Jframe frame = new Jframe("Check");
      frame.setDefaultCloseOperation(Jframe.EXIT_ON_CLOSE);
         Container pane = frame.getContentPane();
         pane.setLayout(new BorderLayout());

         Jpanel canvas = new Jpanel(), panel = new Jpanel();

         panel.add(new Jbutton("One"));
         panel.add(new Jbutton("Two"));

         pane.addMouseListener(new MouseListener() {
                public void mouseClicked(MouseEvent e) {
                   System.out.println(e.getX()+","+e.getY());
                }
                public void mouseEntered(MouseEvent e) {}
                public void mouseExited(MouseEvent e) {}
                public void mousePressed(MouseEvent e) {}
                public void mouseReleased(MouseEvent e) {}
            });

         pane.add(canvas,BorderLayout.CENTER);
         pane.add(panel,BorderLayout.SOUTH);

      frame.setSize(200,140);
      frame.setVisible(true);
   }
}
```

כפי שניתן לראות, מאזין לעכבר צריך לממש מספר פונקציות מכיוון שישנם מספר סוגים של מאורעות שיכולים להתרחש. בדוגמה זו ממשנו רק את הפונקציה האחראית על לחיצה, ואת כל השאר השארנו ריקות.

מכיוון שכתיבת המימושים הריקים, מכבידים על המתכנת ומגדילים את הקוד שלא לצורך, קיימות בספרייה מחלקות מיוחדות החוסכות כתיבה זו. המחלקות עליהן מדובר נקראות Adaptors והן פשוט מחלקות הממשות את כל פונקציות הממשק באופן ריק. כאשר רוצים ליצור מאזין, יורשים ממחלקה כזו וממשים רק את הפונקציות הרלוונטיות.

הדוגמה הבאה מראה שימוש ב Adaptor המיועד למאזיני עכבר - MouseAdapter:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class Check {
   public static void main(String[] args) {
      JFrame frame = new JFrame("Check");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         Container pane = frame.getContentPane();
         pane.setLayout(new BorderLayout());

         JPanel canvas = new JPanel(), panel = new JPanel();

         panel.add(new JButton("One"));
         panel.add(new JButton("Two"));

         pane.addMouseListener(new MouseAdapter() {
                public void mouseClicked(MouseEvent e) {
                   System.out.println(e.getX()+","+e.getY());
                }
            });

         pane.add(canvas,BorderLayout.CENTER);
         pane.add(panel,BorderLayout.SOUTH);

      frame.setSize(200,140);
      frame.setVisible(true);
   }
}
```

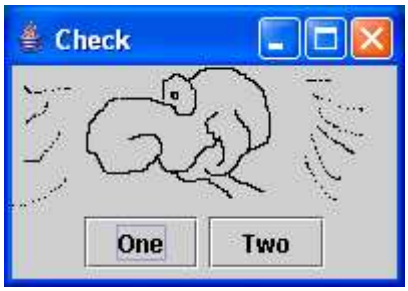
שימוש בגרפיקה

על אובייקט מסוג Graphics אפשר לחשוב כמכשיר ציור משוכלל. מכל קומפוננט גרפי אפשר לבקש מכשיר ציור כזה באמצעות הפונקציה getGraphics(). הקוד הבא מוסיף לתכנית הקודמת את האפשרות לצייר על ה - canvas:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class Check {
   static Graphics _gr;
   public static void main(String[] args) {
      Jframe frame = new Jframe("Check");
      frame.setDefaultCloseOperation(Jframe.EXIT_ON_CLOSE);
         Container pane = frame.getContentPane();
         pane.setLayout(new BorderLayout());

         Jpanel canvas = new Jpanel(), panel = new Jpanel();

         panel.add(new Jbutton("One"));
         panel.add(new Jbutton("Two"));

         canvas.addMouseMotionListener(new MouseMotionAdapter () {
                public void mouseDragged(MouseEvent e) {
```

```
                _gr.drawLine(e.getX(),e.getY(),
                            e.getX(),e.getY());
            }
        });

        pane.add(canvas,BorderLayout.CENTER);
        pane.add(panel,BorderLayout.SOUTH);

    frame.setSize(200,140);
    frame.setVisible(true);
        _gr = canvas.getGraphics();
  }
}
```

דוגמה לפלט:



למרות שהקוד האחרון תקין, הוא איננו אופטימלי מבחינת ארכיטקטורה. בתכנות מונחה עצמים אנו שואפים להגדיר אובייקטים המסוגלים לבצע משימות בעצמם. הקוד שכתבנו מכתיב מבחוץ לאובייקטים, מה עליהם לעשות. לקוד הבא ארכיטקטורה משופרת:

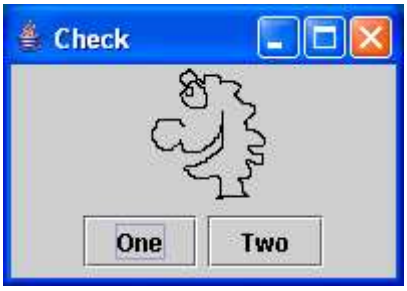
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class Check extends JFrame {
  Graphics _gr;
  public Check() {
    super("Check");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container pane = getContentPane();
        pane.setLayout(new BorderLayout());

        JPanel canvas = new JPanel(), panel = new JPanel();

        panel.add(new JButton("One"));
        panel.add(new JButton("Two"));

        canvas.addMouseMotionListener(new MouseMotionAdapter () {
                public void mouseDragged(MouseEvent e) {
                _gr.drawLine(e.getX(),e.getY(),  // how about "getGraphics().drawLine(…)
                            e.getX(),e.getY());
                }
        });
```

```
        pane.add(canvas,BorderLayout.CENTER);
        pane.add(panel,BorderLayout.SOUTH);

    setSize(200,140);
    setVisible(true);
        _gr = canvas.getGraphics();
  }
  public static void main(String[] args) { new Check(); }
}
```

התוכנית שכתבנו עד כה יכולה לשמש כתוכנית ציור בסיסית ביותר. הציור היה מבוסס על גרירה של העכבר. מכיוון שגרירה כזאת מייצרת מאורעות בהפרשי זמן קבועים, ישנם מקרים בהם נקודות הציור אינן מחוברות. ניתן לשפר את התוכנית ע"י ציור קו בין כל שתי נקודות עוקבות בהן התרחש מאורע גרירה:

```
class Check extends JFrame {
        boolean first=true;
        int _x,_y;
        …

        canvas.addMouseMotionListener(new MouseMotionAdapter () {
                public void mouseDragged(MouseEvent e) {
                  int x = e.getX(), y = e.getY();
                  if(first)
                        first = false;
                  else
                        _gr.drawLine(_x,_y,x,y);
                  _x = x; _y = y;
                }
            });

        …
}
```

פלט אפשרי:



ישנן עוד אפשרויות רבות וכלים רבים המסופקים ע"י החבילות הגרפיות. הדברים שהוצגו כאן יכולים רק לשמש כמבוא וכנקודת התחלה ליצירת ממשקים חלונאיים.