CODESPRING
OUR CODE YOUR SUCCESS
20 years

Codespring mentorprogram
Enterprise szoftverrendszerek fejlesztésének alapjai (MP I)

Let's win the race together!

AWT grafika

Simon Károly
simon.karoly@codespring.ro

Bevezető

- Grafikus felületek a programozó szempontjából: grafikus elemek absztrakt reprezentációja $\longrightarrow$ az egyes elemek tulajdonságait leíró adatstruktúrák.
- Fogalmak: geometriai alakzatok, felületek, színek, textúrák, fények, pozíciók, méretek, betűtípusok stb.
- Amit a felhasználó lát: a kimeneti eszközön (általában képernyőn) megjelenő kép, amely tulajdonképpen különböző színű pixelekből áll.
- A kép a modell alapján készül, de konkrét megjelenítését más tényezők is befolyásolják (kimeneti eszköz tulajdonságai, pl. felbontás stb.).
- Renderelés (rendering): a folyamat, amely a belső reprezentációt a megjelenítendő képbe képezi
- Egy adott modell alapján létrehoz egy digitális képet, bittérképet (raster image, bitmap)
- A modell leképezését egy natív mechanizmus biztosítja (mivel a megjelenítés platformfüggő)

Bevezető

- Pl. gomb (a megfelelő tulajdonságokkal) → háttérszínű pixelekkel kitöltött téglalap alakú felület, amelyen ír valamit (a felbontásnak megfelelően megjelenítve).
- Általában elegendő, ha az eszköztárak által biztosított standard komponenseket használjuk, de megtörténhet, hogy speciálisabb megjelenítésű komponenseket akarunk létrehozni (pl. grafikus elemeket elhelyezve egy komponens felületén).
- Az AWT eszközkészlet esetében a vászon (Canvas) komponens segítségével teljesen egyedi grafikus felületeket hozhatunk létre.
- A Java több lehetőséget is biztosít grafikus elemek létrehozására: Java 2D API, Java 3D API, JOGL, JMonkey stb.

A Graphics osztály

- `Graphics` absztrakt alaposztály: lehetővé teszi az alkalmazások számára, hogy különböző komponensek felületére rajzoljanak (a grafikus modell reprezentációja).
- A Java által támogatott alapvető renderelési műveletek elvégzéséhez szükséges információkat, tulajdonságokat tárolja (forma, szín, betűtípus, méret, pozíció, stb.), valamint metódusokat tartalmaz ezek beállítására, és különböző alakzatok kirajzolására.
- Az osztály (és metódusainak többsége) absztrakt. A renderelés folyamata platform-specifikus, natív kód segítségével történhet. A különböző platformokra írt JVM-ek különböző implementációkat biztosítanak az osztálynak, natív mechanizmusokat alkalmazva a platform ablakrendszerével történő „együttműködésre”.

A Graphics osztály

- Néhány példa:

```
void setColor(Color c)
void drawRect(int x, int y, int width, int height)
void fillRect(int x, int y, int width, int height)
```

- Továbbá: **`setFont`**, **`drawOval`**, **`drawPolygon`**, **`drawString`**, **`drawImage`** stb.
- Közvetlen leszármazottja a Graphics2D (szintén absztrakt), amely további lehetőségeket biztosít (pl. hatékonyabb módszereket a színkezelésre, transzformációkra stb.)

AWT komponensek megjelenítése és frissítése

- A rendszer kezdeményezésére (system-triggered painting): az ablakrendszer kéri az illető komponens frissítését. Ez akkor fordul elő, amikor a komponens először válik láthatóvá, újraméreteződik, vagy valamilyen okból „sérül” a felülete (pl. föléje helyezünk, majd eltávolítunk egy másik komponenst, vagy lekicsinyítjük, majd visszaállítjuk a komponenst tartalmazó ablakot).
- Az alkalmazás kezdeményezésére (application-triggered painting): az alkalmazás valamilyen belső állapotváltozás következményeként úgy dönt, hogy a komponens felülete frissítésre szorul.
- A komponensek felületének kirajzolása visszahívásos mechanizmuson (callback) alapszik: a renderelésért felelős programrészt egy adott metóduson belül kell elhelyezni, és ezt a metódust hívja meg a toolkit a komponens felületének kirajzolásakor, vagy frissítésekor. Ez a metódus a `Component` osztály `paint` metódusa, melyet a különböző komponensek a nekik megfelelő módon újradefiniálnak.

Paint


```
public void paint(Graphics g)
```

- A paraméterként kapott **`Graphics`** objektum a komponens felületének grafikus modellje. A paraméter határozza meg a grafikus megjelenítéssel kapcsolatos tulajdonságokat (szín, betűtípus stb.), illetve a frissítendő felületet (azt a részt, amely újra lesz renderelve) is behatárolja (clipping).
- A tulajdonságok a paint metóduson belül módosíthatóak, a módosítások, illetve a rajzolási műveletek a **`g`** referencia segítségével történhetnek.
- A rendszer által kezdeményezett frissítés esetében a rendszer behatárolja a „sérült” részt (amely a teljes felület is lehet), és meghívja a paint metódust. A rendszer feltételezi, hogy a sérült felület teljes frissítésre szorul, és minden pixelt frissít (tulajdonképpen törli és újrarajzolja a felületet).
- Az alkalmazás általi frissítés kezdeményezése a **`repaint`** metódus meghívásával történik. A repaint egy a komponens felületének frissítésére vonatkozó aszinkron kérés a toolkit-nek címezve. A metódus meghívása nem vezet azonnali paint metódushíváshoz. A kérés teljesítésekor a toolkit először meghívja a komponens **`update`** metódusát.

Repaint, update


```
public void update(Graphics g)
```

- A metódus alapértelmezett implementációja törli a felületet és meghívja a paint metódust, tehát teljesen újrarajzolja az érintett felületet, az „üresen” maradt részeket háttérszínű pixelekkel töltve ki.
- Az alapvető különbség az előbbi esethez képest (rendszer által kezdeményezett frissítés) abból ered, hogy az update újradefiniálható. A mechanizmus lehetőséget ad arra, hogy ne feltétlenül kelljen törölnünk az érintett felületet, és ilyen módon egy már kirajzolt részt új grafikus elemekkel egészíthetünk ki (incremental painting).
- A **`repaint`** metódusnak különböző változatai vannak: meghívhatjuk paraméterek nélkül (a komponens teljes felületének frissítésekor), paraméterek segítségével behatárolhatjuk a frissítendő részt, illetve megadhatunk egy „határidőt” (mennyi időn belül következzen a paint hívás).

Repaint, update

- A repaint eredménye egy aszinkron kérés, amely nem feltétlenül vezet azonnali update/paint híváshoz. Megtörténhet, hogy mielőtt az első kérésnek megfelelő paint metódushívás megtörténne, további kérések érkeznek. Az ilyen esetekben a több kérés egyetlen paint hívásba lesz összevonva. Ez a mechanizmus különösen hasznos lehet akkor, amikor az alkalmazáson belül egymás után (például egy cikluson belül) több apró frissítést szeretnénk elvégezni.
- Az alkalmazás által kezdeményezett frissítéseknél sohasem hívjuk meg közvetlen módon a paint metódust. A frissítést minden esetben a repaint metódushíváson keresztül kérjük. Ez természetes is, tudva azt, hogy a paint paraméterének típusa Graphics, és absztrakt osztályból nem példányosíthatunk. A paint direkt módon történő meghívása legfeljebb az újradefiniált update metóduson belül történhet (továbbadva a metódusnak az update paraméterét).
- A grafikus komponensek mindegyike újradefiniálja a neki megfelelő módon a paint metódust. Ha valami speciális megjelenítést szeretnénk, akkor ezt a származtatott osztályainkban mi is megtehetjük. Ha csak ki szeretnénk egészíteni a komponensek grafikus tartalmát, akkor először a paint-en belül a super referencia segítségével meghívjuk az alaposztály paint metódusát, és ezt követhetik a saját utasítások (ez különösen hasznos lehet tárolók megfelelő megjelenítésének esetében). Ha egyszerűen csak rajzolni szeretnénk, akkor használhatjuk a rajzvászon (Canvas) osztályt, ebből származtatva saját komponensünket, felülírva a paint metódust.

Vászon - példa

- Egy kereten belül helyezzünk el egy rajzvásznat, és adjunk lehetőséget arra, hogy a felhasználó az egér segítségével kis köröket rajzolhasson ki erre a vászonra.

```
import java.awt.BorderLayout;
import java.awt.Canvas;
import java.awt.Color;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.Graphics;
import java.awt.Frame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class MyCanvas extends Canvas {

  private int x = 0;
  private int y = 0;

  public MyCanvas() {
    setBackground(new Color(50,100,250));
    addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent e) {
        x = e.getX();
        y = e.getY();
        repaint();
      }
    });
  }
```

Vászon - példa


```
    public void paint(Graphics g) {
        g.setColor(Color.red);
        g.fillOval(x,y,20,20);
    }

    public static void main(String args[]) {
        Frame f = new Frame("Paint");
        f.setBounds(50,50,300,200);
        f.add(new MyCanvas(), BorderLayout.CENTER);
        f.addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent e) {
              System.exit(0);
          }
        });
        f.setVisible(true);
    }

}
```

Vászon - példa

- Ha többször is kattintunk, mindig csak az utolsó kör lesz látható, az előző eltűnik. Ez azért van így, mert a teljes felület frissítését kérjük, és az **update** alapértelmezett implementációja törli a felületet a **paint** meghívása előtt. Ha azt szeretnénk, hogy az előzőekben kirajzolt körök is láthatóak maradjanak, egyszerűen kiegészítjük az osztályunkat, újradefiniálva az update metódust:

```
public void update(Graphics g) {
  paint(g);
}
```

- Az újradefiniált **update** metóduson belül egyszerűen meghívjuk a **paint** metódust, nem végzünk el törlési műveletet, így az előzőleg kirajzolt grafikus elemek láthatóak maradnak.

Vászon - példa

- Ha a rendszer kéri a felület újrarajzolását, például, mikor lekicsinyítjük az ablakot, majd visszaállítjuk eredeti méreteit, ismét csak a legutolsó kör lesz látható. Ilyen esetben a rendszer a teljes felület frissítését kéri, és nem tárolja az előzőleg megjelenített körök koordinátáit. Természetesen a koordinátákat rögzíthettük volna például egy listában, eszerint módosítva a `paint` metódust, de van ennél elegánsabb megoldás is.
- Egy kép objektumot alkalmazhatunk. A köröcskéket erre a képre rajzoljuk rá, és a `paint` metóduson belül a képet rajzoljuk ki a vászonra, így az előző módosításokat is megőrizhetjük.
- Digitális képek (bittérképek) létrehozásában és kezelésében az `Image` absztrakt alaposztály lehet segítségünkre. Minden olyan osztály, amely egy grafikus kép reprezentációja ennek az osztálynak a leszármazottja.
- Lévén absztrakt alaposztályról szó, közvetlen módon nem példányosíthatunk belőle, de több módszer is van kép objektum létrehozására. Például egyszerűen példányosíthatunk a `BufferedImage` származtatott osztályból. Másik lehetőség: a `Component` osztály `createImage` metódusa

Vászon - példa

- A `createImage` metódus eredetileg a kettős pufferelés (double buffering) mechanizmus támogatásának céljából kapott helyet a `Component` osztályban.
  - A kettős (vagy általánosabban többszörös) pufferelés mechanizmusát a számítógépes grafikában a megjelenítés optimalizálására, a képfrissítés gyorsítására alkalmazzák. A mechanizmus lényege, hogy az új kép létrehozásakor nem közvetlenül a videomemóriával dolgozunk. A rajzolási műveletek eredményeit előzőleg a memóriában, egy háttér pufferben tároljuk, majd amikor elkészült a teljes kép, ennek a puffernek a tartalmát egy gyors művelettel a videomemóriába másoljuk, lecserélve az aktuálisan látható képet a háttérben elkészített képre. Ilyen módon felgyorsítható a képfrissítés, és elkerülhetőek az olyan kellemetlenségek, mint a kép villogása a folyamatos rajzolási műveletek miatt, vagy a régi és új grafikus elemek keveredése a megjelenített képen belül a frissítés során. A createImage metódus által visszatérített Image objektum ilyen háttér pufferként szolgálhat, segítségével a háttérben elvégezhetőek a rajzolási műveletek a felület frissítése, az új kép megjelenítése előtt.
- Fontos megkötés: a komponensnek láthatónak kell lennie, ellenkező esetben a metódus `null` értéket ad.
  - A megkötésnek egyszerűen megfelelhetünk, ha a képet a paint metóduson belül hozzuk létre (a metódus meghívásakor a komponensünk már biztosan látható).

Vászon - példa

- Hogyan rajzolhatunk rá a képre? Hasonlóan, mint ahogyan azt a komponens felületének esetében is tettük, a **`Graphics`** osztály segítségével. Az **`Image`** osztályokhoz tartozik egy **`Graphics`** objektum, és a **`getGraphics`** metódus segítségével kérhető egy erre mutató referencia. A rajzolási műveleteket a referencia segítségével végezhetjük. Magát a képet a komponens felületére fogjuk kirajzolni a **`paint`** metódus paramétereként kapott **`Graphics`** típusú objektumra meghívva a **`drawImage`** metódust.
- Ezekkel a módosításokkal elérjük, hogy az előzőleg kirajzolt körök is megmaradjanak, még akkor is, ha a rendszer kéri a felület frissítését.
  - „Szépséghiba”: ha a kép méreteit a rajzvászon méreteinek segítségével adjuk meg, és a keret újraméretezésénél ezt nem változtatjuk, majd kinagyítjuk a keretet, a rajzvászon bizonyos részeire nem tudunk rajzolni. Persze, a kerettel együtt újraméretezhetjük a képet is. Ebben az esetben torzulhatna a kép, de ez is könnyen kikerülhető (a legegyszerűbb már először egy nagyobb képet létrehozni, de megoldhatjuk úgy is a problémát, hogy a nagyobb kép létrehozásakor rárajzoljuk az előbbi kisebb képet).

Vászon - példa


```
... //az import utasítások
public class MyCanvas extends Canvas {
  private Image img;
  private Graphics gr;

  public MyCanvas() {
    setBackground(new Color(50,100,250));
    addMouseListener(new MouseAdapter(){
      public void mousePressed(MouseEvent e){
        gr.fillOval(e.getX(),e.getY(),20,20);
        repaint();
      }
    });
  }

  public void paint(Graphics g) {
    if (img == null) {
      img = createImage(getWidth(),getHeight());
      gr = img.getGraphics();
      gr.setColor(Color.red);
    }
    g.drawImage(img,0,0,null);
}
public void update(Graphics g) {
    paint(g);
}

... //a main metódus
}
```

Vászon - példa

- A kép létrehozásához alkalmazott módszer, nem minden esetben a legmegfelelőbb. A használatával kapcsolatos megkötésről már szóltunk, de megemlíthetjük azt is, hogy az ilyen módon létrehozott kép objektum nem tartalmazhat átlátszó pixeleket. Szerencsére, több módszer is rendelkezésünkre áll.
- Az új kép létrehozása helyett állományból is betölthettünk volna egy képet, és ugyanígy le is menthetjük az „alkotásunkat". Erre is több lehetőség van, de közülük kiemelhetjük a **`javax.imageio`** csomag, illetve az ezen belül található **`ImageIO`** osztály használatát.
- A SWING eszközkészlet használata a grafika szempontjából (is) eltéréseket mutat az AWT csomaghoz képest. Ezekre a SWING eszköztárral kapcsolatos részben térünk ki.

Feladat

- Egy kereten belül helyezzünk el egy rajzvásznat (egy saját osztályt hozunk létre a `Canvas` osztályból származtatva, két Choice komponenst, egy jelölőnégyzetet (`Checkbox`), és egy gombot. A felhasználó a két `Choice` komponens segítségével kiválaszthat egy adott alakzattípust (pl. kör, négyzet stb.) és egy adott színt (pl. kék, piros stb.). A gomb lenyomásának hatására a vászonra kirajzoljuk a kiválasztott alakzatot a kiválasztott színnel. Amennyiben a jelölőnégyzet be van jelölve, az alakzat felületét is kitöltjük az illető színnel.
- A keretnek megfelelő osztályt (a `Frame` leszármazottja), és a rajzvászonnak megfelelő osztályt (a `Canvas` leszármazottja) külön osztályként, külön állományokban hozzuk létre (a vásznat ne belső osztályként valósítsuk meg. Figyeljünk arra, hogy a vászon ne függjön a kereten belül alkalmazott komponensektől (pl. ne befolyásolja a vászon osztályt, ha valamelyik `Choice` komponenst listára cseréljük stb.)
- A programnak elkészíthetjük egy olyan változatát is, amelynek esetében nem szükséges a gomb lenyomása: bármelyik másik komponens állapotának változásakor frissítjük a rajzot. Ezen kívül a szín kiválasztására alkalmas `Choice` komponenst helyettesíthetjük olyan módon, hogy a felhasználó tetszőleges R, G, B értékeket meg tudjon határozni (pl. három szövegmező segítségével).