Rapport: Uppgift 3 - Cirkelgång
Namn: Ludvig Fendert
E-post: l.fendert@gmail.com

Inledning

Jag har skapat ett Java-program som ritar och hanterar cirklar i ett grafiskt användargränssnitt (GUI) med hjälp av Swing och AWT. Programmet gör det möjligt för användaren att:

1. Välja färg för nya cirklar genom att klicka på färgrutor i övre vänstra hörnet.
2. Skapa nya cirklar genom att klicka på en tom plats i fönstret.
3. Flytta cirklar genom att klicka inuti en cirkel och dra med musen.
4. Ändra storlek på cirklar genom att klicka på cirkelns kant och sedan dra.

Uppgiften krävde även att den senast flyttade eller skapade cirkeln ska ligga “överst” ifall cirklarna överlappar.

Programstruktur

Programmet består av tre centrala klasser:

1. Circle
    - Syfte: Representerar en cirkel med egenskaper som x- och y-koordinater för centrum, radie och färg.
    - Viktiga metoder:
        - draw(Graphics g): Ritar cirkeln på den givna Graphics-ytan.
        - contains(int mx, int my): Returnerar true om en viss musposition är inuti cirkeln.

        - onEdge(int mx, int my): Returnerar true om muspositionen är nära cirkelns kant (inom en viss tolerans).
        - Getters och setters för t.ex. x, y och radius.

2. CirclePanel (ärver av JPanel)
    - Syfte: Hanterar all interaktion med cirklarna, d.v.s. klick, drag, färgval och uppritning.
    - Viktiga metoder och fält:
        - paintComponent(Graphics g): Ritar både färgrutorna för färgval och alla cirklar.
        - ArrayList<Circle> circles: En lista över alla cirklar i panelen.
        - mousePressed(MouseEvent e), mouseDragged(MouseEvent e), mouseReleased(MouseEvent e):
            - Identifierar om man klickar på en färgruta eller en cirkel.
            - Sätter läget till “drag” eller “resize” beroende på var man klickade (innanför cirkeln eller på kanten).
            - Flyttar eller ändrar storlek på den aktiva cirkeln under mouseDragged().
        - currentColor: Håller koll på vilken färg som är vald för nya cirklar.

3. CircleFrame (ärver av JFrame)
    - Syfte: Fungerar som huvudfönster för programmet. Här skapas ett CirclePanel som läggs till i ramen.
    - Viktig metod:
        - main(String[] args): Programstart. Skapar en instans av CircleFrame som i sin tur skapar CirclePanel.

Hur jag kom fram till min lösning

1. Start med enkel cirkelritning: Jag började med att rita upp en enkel cirkel i en panel (CirclePanel) och se till att den målades om när fönstret uppdateras (paintComponent).

2. Interaktivitet via MouseListener och MouseMotionListener: Jag lade till funktioner för att hantera klick och dra:

    - Först hanterade jag bara “drag-läge”, d.v.s. att klicka inuti cirkeln och dra den.

    - Därefter utökade jag koden för att även stödja “resize-läge” genom att klicka på kanten.

3. Överlapp och cirklar i olika lager: För att cirklarna ska kunna ligga “överst” när man klickar på dem, tog jag bort den cirkel som klickats på ur listan och lade till den sist igen. Därmed ritas den sist och hamnar visuellt överst.

4. Färgval via klick på färgrutor: Jag ritade fyra små rektanglar i hörnet (colorBoxes) och kontrollerar i mousePressed() om klicket hamnar i någon av dessa rektanglar. Om ja, uppdateras currentColor.

Jag har i stor utsträckning utnyttjat standardbibliotekets Swing- och AWT-klasser:

- JFrame, JPanel för GUI-uppbyggnad,

- Color, Graphics för färg och ritning,

- MouseListener, MouseMotionListener för att lyssna på musinteraktion.

Designbeslut och motivering

1. En klass för cirklar (Circle)

    - Jag ville hålla cirkelns data och logik (t.ex. koll av klick på kanten) utanför själva GUI-koden. Därför skapade jag en fristående Circle-klass som inte

ärver av Swing-klasser.

2. Panel för interaktion (CirclePanel)
    - Genom att lägga all logik för att hantera musen i en enda panel är det enkelt att se hur cirklarna skapas, flyttas och ritas.
3. Ingen “storlek vid skapande”
    - Jag valde att inte implementera möjligheten att klicka-dra för att direkt bestämma cirkelns radie när den skapas. Istället skapas en cirkel med fast radie (30 px) och man kan sedan ändra storleken. Detta gör koden enklare och uppfyller uppgiftens krav.
4. Simpel datastruktur (ArrayList)
    - Jag använder en ArrayList<Circle> för att lagra cirklarna. Det är enkelt att iterera över den i omvänd ordning när man letar efter vilken cirkel som är “överst” och som träffas av ett musklick.

Utmaningar och lösningar

- Överlappande cirklar: När två cirklar överlappar måste klick på den översta cirkeln prioriteras. Lösningen är att iterera över listan baklänges (från sist skapad till först skapad) när jag kollar contains() eller onEdge(). Den första träffade cirkeln blir den aktiva.
- Flytthopp: Jag ville undvika att cirkeln “hoppar” till muspekarens position när man börjar dra. Därför beräknar jag ett offset (offsetX och offsetY) i mousePressed() och använder det sedan i mouseDragged() för att bibehålla avståndet mellan cirkelns centrum och muspekarens position.
- Storleksändring: För att ändra cirkelns storlek räknar jag helt enkelt ut avståndet mellan muspekarens position och cirkelns centrum med Pythagoras sats och sätter det som cirkelns nya radie.

Efterhandsutvärdering

Överlag är jag nöjd med min design. Den är relativt enkel, tydlig och uppfyller alla krav. Några förbättringspunkter skulle kunna vara:

1. Direkt storleksval vid skapande: Att låta användaren klicka och dra för att “dra ut” en cirkel första gången skulle vara mer intuitivt.

2. Visuell feedback: Det hade varit trevligt att byta muspekare när man är på kanten (t.ex. till en “resize”-ikon) eller när man är inuti en cirkel (t.ex. till en “move”-ikon).

3. Större design för färgval: Just nu är färgrutorna hårdkodade med fyra färger. Man hade kunnat ha en mer avancerad färgpalett eller en färgväljare (JColorChooser).

Trots detta uppfyller programmet alla krav och fungerar bra för uppgiften.

Möjliga förbättringar

- Förbättrad visuell feedback:
  Att byta muspekare beroende på interaktion kan göra användarupplevelsen mer intuitiv. Exempelvis kan man visa en “resize”-ikon när musen är nära cirkelns kant och en “move”-ikon när den är inuti cirkeln.

- Direkt storleksval vid skapande:
  Istället för att alltid skapa en ny cirkel med en fast radie (30 px) och sedan låta användaren ändra storlek, kan man implementera en mekanism där användaren klickar och drar för att direkt definiera cirkelns storlek vid skapandet.

- Utökat färgval:
  Färgrutorna är för närvarande hårdkodade med fyra färger. En förbättring skulle vara att använda en mer flexibel färgpalett, exempelvis med en färgväljare som JColorChooser, vilket ger användaren möjlighet att välja en bredare variation av färger.

Sammanfattning

Mitt program “Cirkelgång” ritar och hanterar cirklar i ett Swing-fönster. Med hjälp av MouseListener och MouseMotionListener kan användaren skapa, flytta och ändra storlek på cirklar samt välja färg för nya cirklar. Koden är uppdelad i tre klasser (Circle, CirclePanel och CircleFrame), där ansvaret för cirkelns data och ritning ligger i Circle, medan CirclePanel hanterar användarinteraktionen. Denna uppdelning gör koden överskådlig och relativt enkel att vidareutveckla eller modifiera.

Teknisk information

IDE: IntelliJ

javac --version
java --version

javac 22.0.1
java 22.0.1 2024-04-16

Java(TM) SE Runtime Environment (build 22.0.1+8-16)
Java HotSpot(TM) 64-Bit Server VM (build 22.0.1+8-16, mixed mode, sharing)
