
# 2D-Graphics


# Java2D

- Graphics, Text, Imaging
- Java2D är en del av Java Foundation Classes som är en integrerad del i Java 2
- Java2D och Swing ersätter/kompletterar AWT
- AWT finns kvar för bakåtkompabilitet
- Java2D baseras där så är möjligt på AWT
- Nackdel: Hastigheten

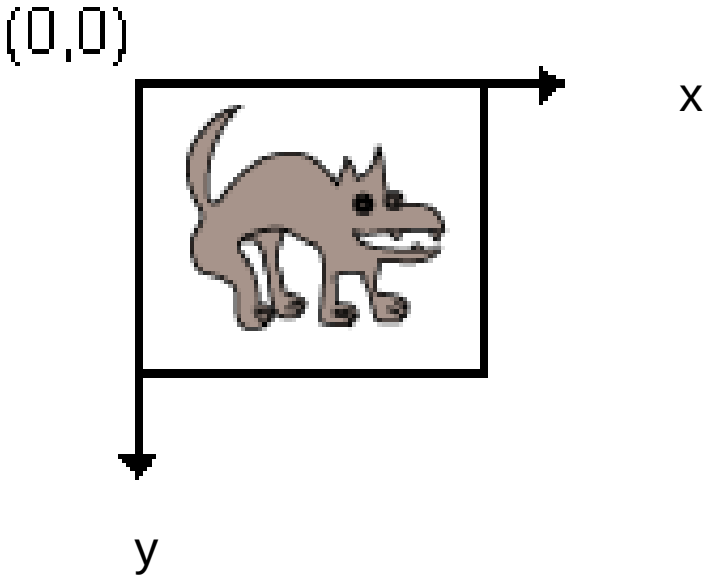

# Koordinatsystem

- Java 2D API maintains two coordinate systems:
  - *User space* is a device independent, logical coordinate system. Applications use this coordinate system exclusively; all geometries passed into Java 2D rendering routines are specified in the user space.
  - *Device space* is a device-dependent coordinate system that varies according to the target rendering device.


# Koordinatsystem

The Java 2D graphics automatically performs the necessary conversions between user space and the device space of the rendering device.

# User space




# User space

The user space represents a uniform abstraction of all possible device coordinate systems. The device space for a particular device might have the same origin and direction as user space, or it might be different.


# Device space

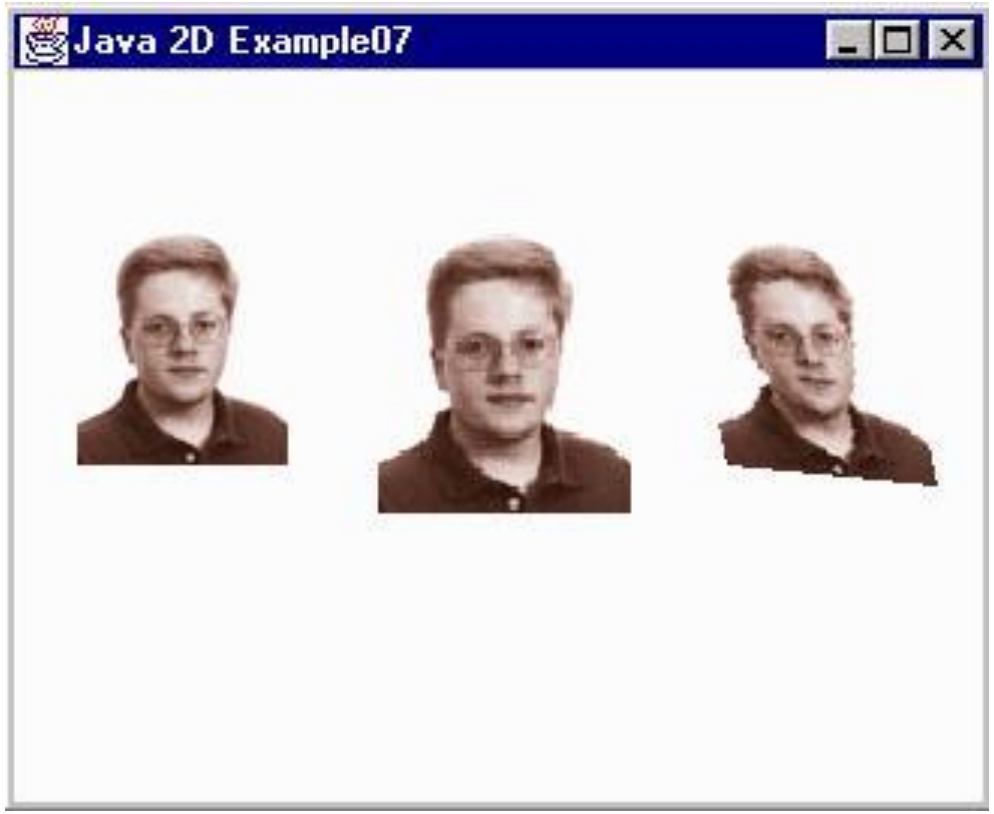
- The Java 2D API defines three levels of configuration information. This information is encapsulated by three classes:
  - *GraphicsEnvironment* describes a collection of all rendering devices on a particular platform (screens, printers, image buffers, fonts).Can contain one ore more GraphicsDevices.
  - *GraphicsDevice* describes an application rendering device (screen, printer). Can have one ore more GraphicsConfiguration.
  - *GraphicsConfiguration*


- Graphics2D utökar AWT-klassen Graphics
- ”User space” är oberoende av device (bildskärm, skrivare, …)
- Koordinater mäts i flyttal (float eller double)
- En längdenhet är en point (1/72 tum) eller ungefär en pixel


# Transformationer

- Koordinatsystemet går att transformera
- Axlarna kan byta riktning och längd
- Exempel på fördefinierade transformationer:
  - Translation
  - Rotation
  - Skalning
  - Skevning
  - Affina transformationer (linjärkombination av gamla basvektorer plus en konstant)

# Transformationer




Three images of Bill in various states of transformation.


# Shapes

- Alla objekt som hanteras i Java2D är ab super-typen Shape
- Shape är ett interface
- En Shape kan vara en linje, en rektangel, en cirkel, en kurva eller en bokstav ur ett typsnitt
- En Shape kan definieras en gång och ritas flera gånger, eventuellt i en transformerad version.


# Graphics context

- Eftersom Java2D är objektbaserat så är definitionen av en kontur (Shape) separerat från själva ritandet.
- Först definierar man sin Shape, sedan ritar man den via en Graphics2D
- Det går både att rita konturer och fylla i objekt
- Färg, linjetjocklek, transparens m.m. hålls reda på i en så kallad ”Graphics context”
- För att rita ett objekt måste man alltså göra tre saker:
  1. Definiera konturerna för ett objekt genom att skapa en shape
  2. Bestäm hur den skall ritas genom att ändra i ”Graphics context” för aktuell Graphics2D
  3. Rita genom att anropa en ritmetod ur Graphics2D med denna Shape som argument

# Graphics context


The *pen style* that is applied to the outline of a shape. This *stroke* attribute enables you to draw lines with any point size and dashing pattern and to apply end-cap and join decorations to a line.


The *fill style* that is applied to a shape's interior. This *paint* attribute enables you to fill shapes with solid colors, gradients, and patterns.


The *compositing style* that is used when rendered objects overlap existing objects.

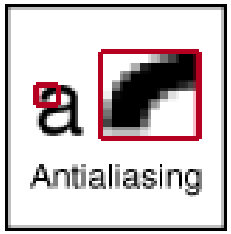
# Graphics context


The *transform* that is applied during rendering to convert the rendered object from user space to device-space coordinates. Optional translation, rotation, scaling, or shearing transforms can also be applied through this attribute.


The *clip*, which restricts rendering to the area within the outline of the Shape used to define the clipping path. Any Shape can be used to define the clip.


The *font* used to convert text strings to glyphs.

# Graphics context




*Rendering hints* that specify preferences in the trade-offs between speed and quality. For example, you can specify whether antialiasing should be used, if it's available.


# AWT - Swing

- AWT
  - public void paint(Graphics g)
- Swing
  - public void paintComponent(Graphics g)
  - public void paintChildren(Graphics g)
  - public void paintBorder(Graphics g)


# AWT - Swing

- **System-triggered Painting**
- In a system-triggered painting operation, the system requests a component to render its contents, usually for one of the following reasons:
- The component is first made visible on the screen.
- The component is resized.
- The component has damage that needs to be repaired. (For example, something that previously obscured the component has moved, and a previously obscured portion of the component has become exposed).
- **App-triggered Painting**
- In an application-triggered painting operation, the component decides it needs to update its contents because its internal state has changed. (For example,. a button detects that a mouse button has been pressed and determines that it needs to paint a "depressed" button visual).

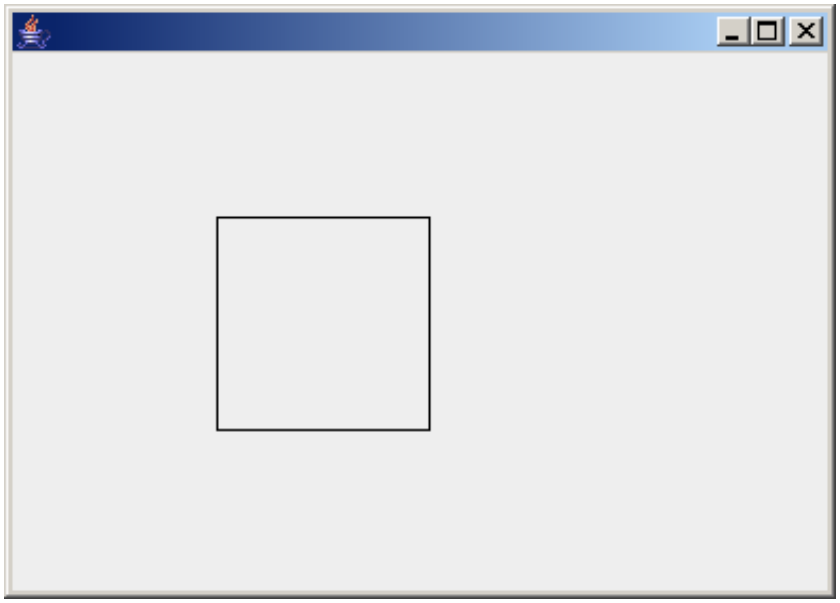

# AWT - Swing

- AWT
  - invalidate() Invalidates this component. This component and all parents above it are marked as needing to be laid out. This method can be called often, so it needs to execute quickly.
  - validate() Ensures that this component has a valid layout. This method is primarily intended to operate on instances of Container.


# MyFrame.java

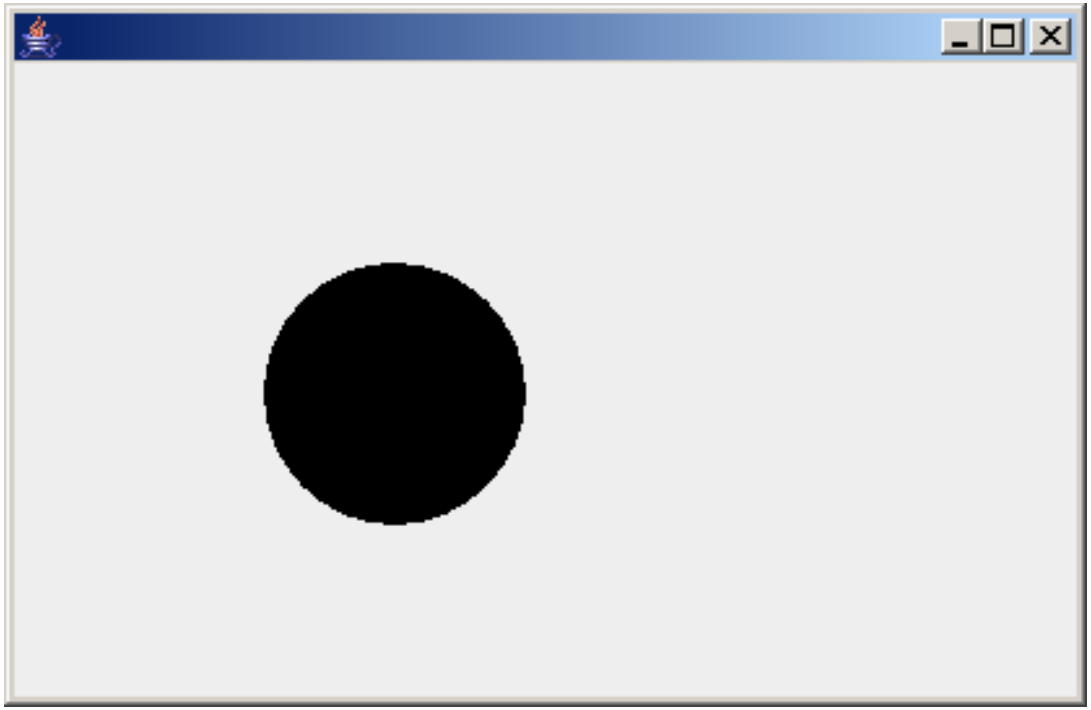
```
import java.awt.Dimension;
import java.awt.Toolkit;

import javax.swing.JFrame;

public class MyFrame extends JFrame
{	MyFrame()
	{	setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		Toolkit tk = Toolkit.getDefaultToolkit();
		Dimension d = tk.getScreenSize();
		int screenHeight = d.height;
		int screenWidth = d.width;
		setSize(screenWidth/2,screenHeight/2);
		setLocation(screenWidth/4, screenHeight/4);
		// setVisible(true);  // why not here?
	}
}
```

# One.java - draw

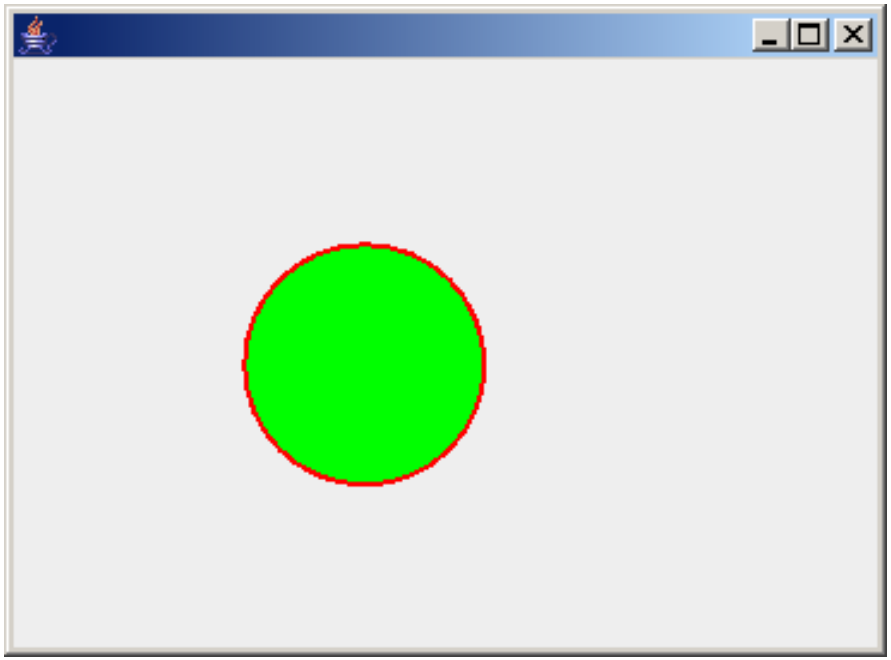


shp=java.awt.geom.Rectangle2D$Double[x=100.0,y=100.0,w=100.0,h=100.0]
--> paint(.)
--> paint(.)
--> paint(.)
--> paint(.)
--> paint(.)

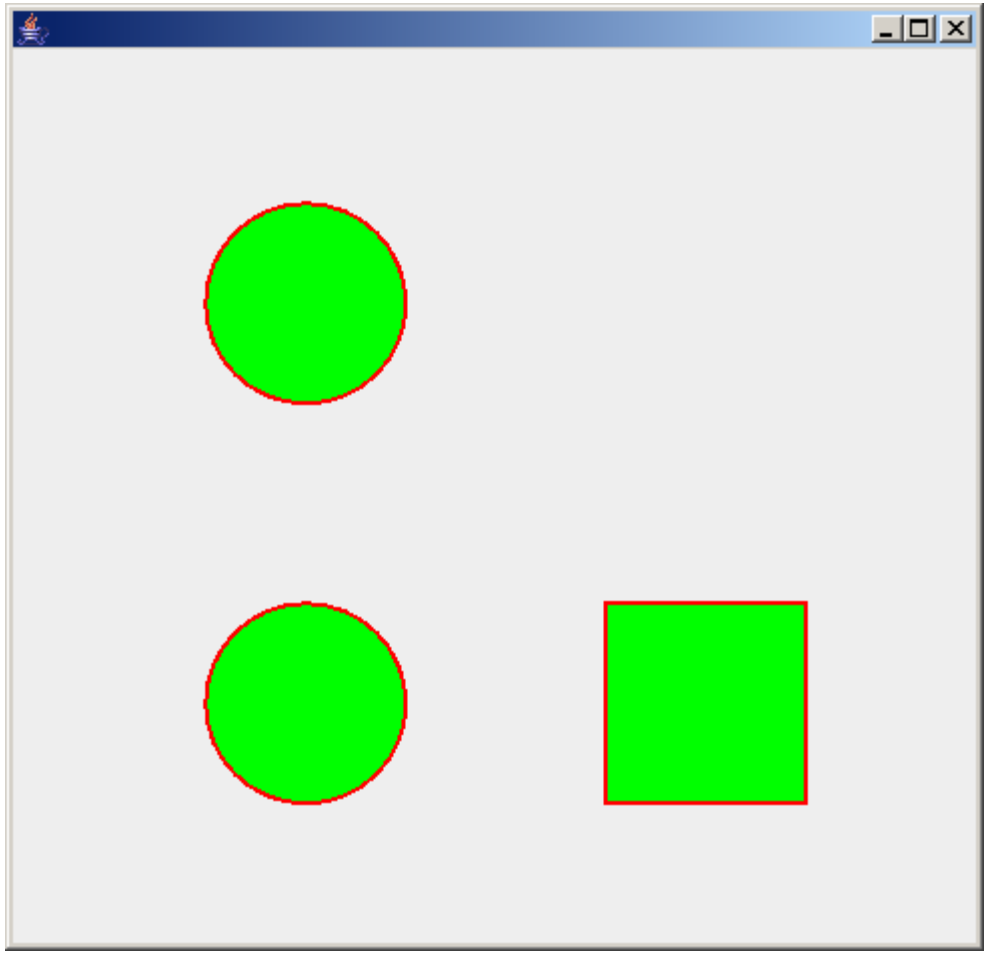
# Two.java – draw, fill



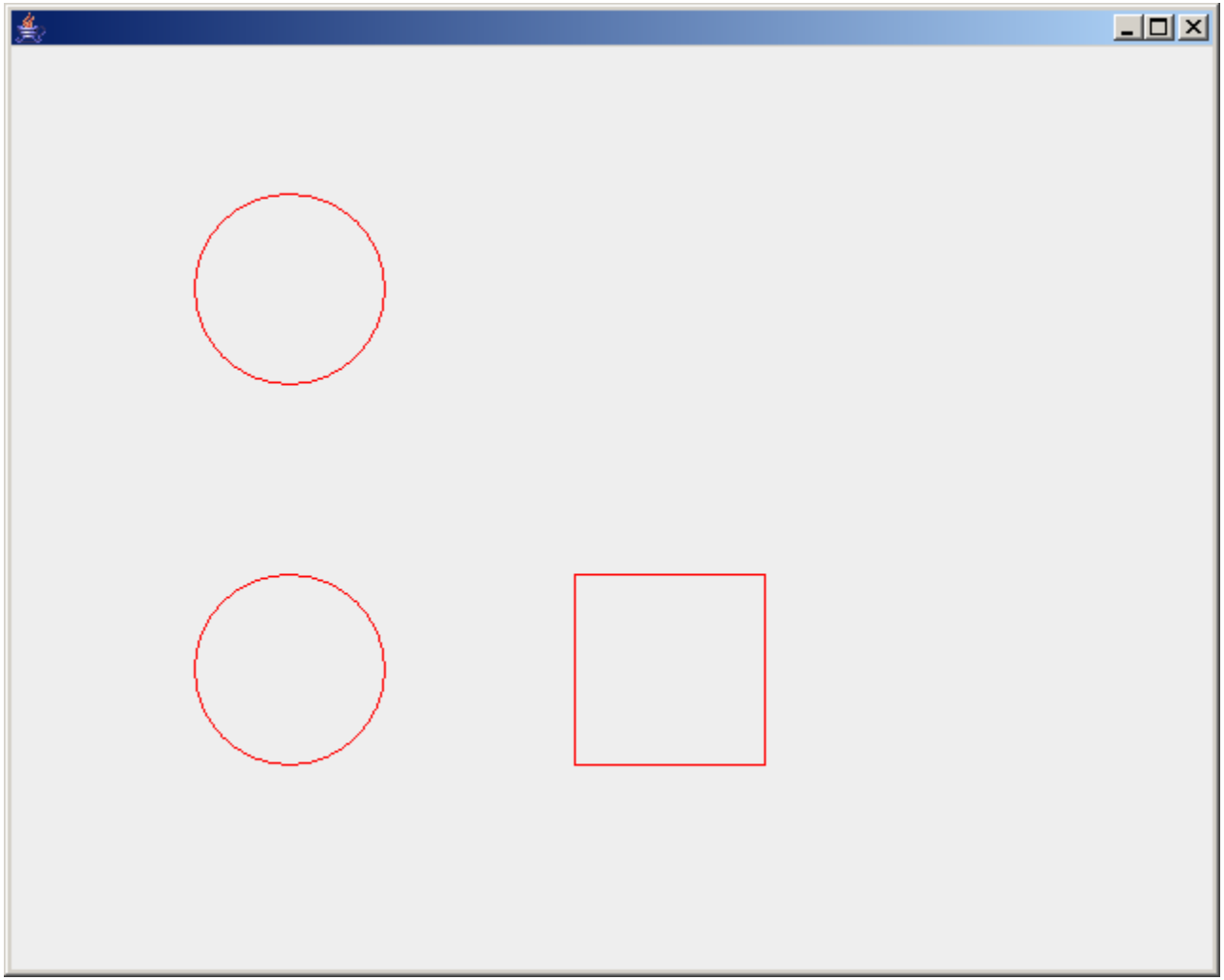

# Three.java – set, draw, fill



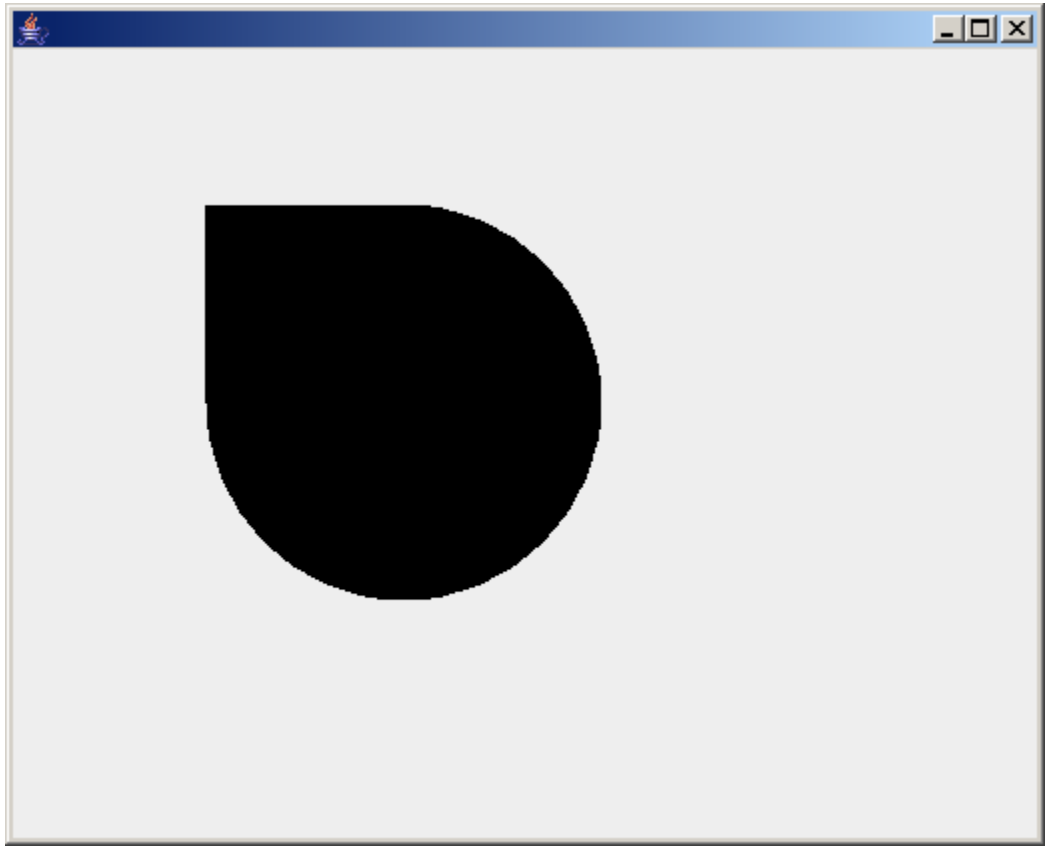

# Four.java



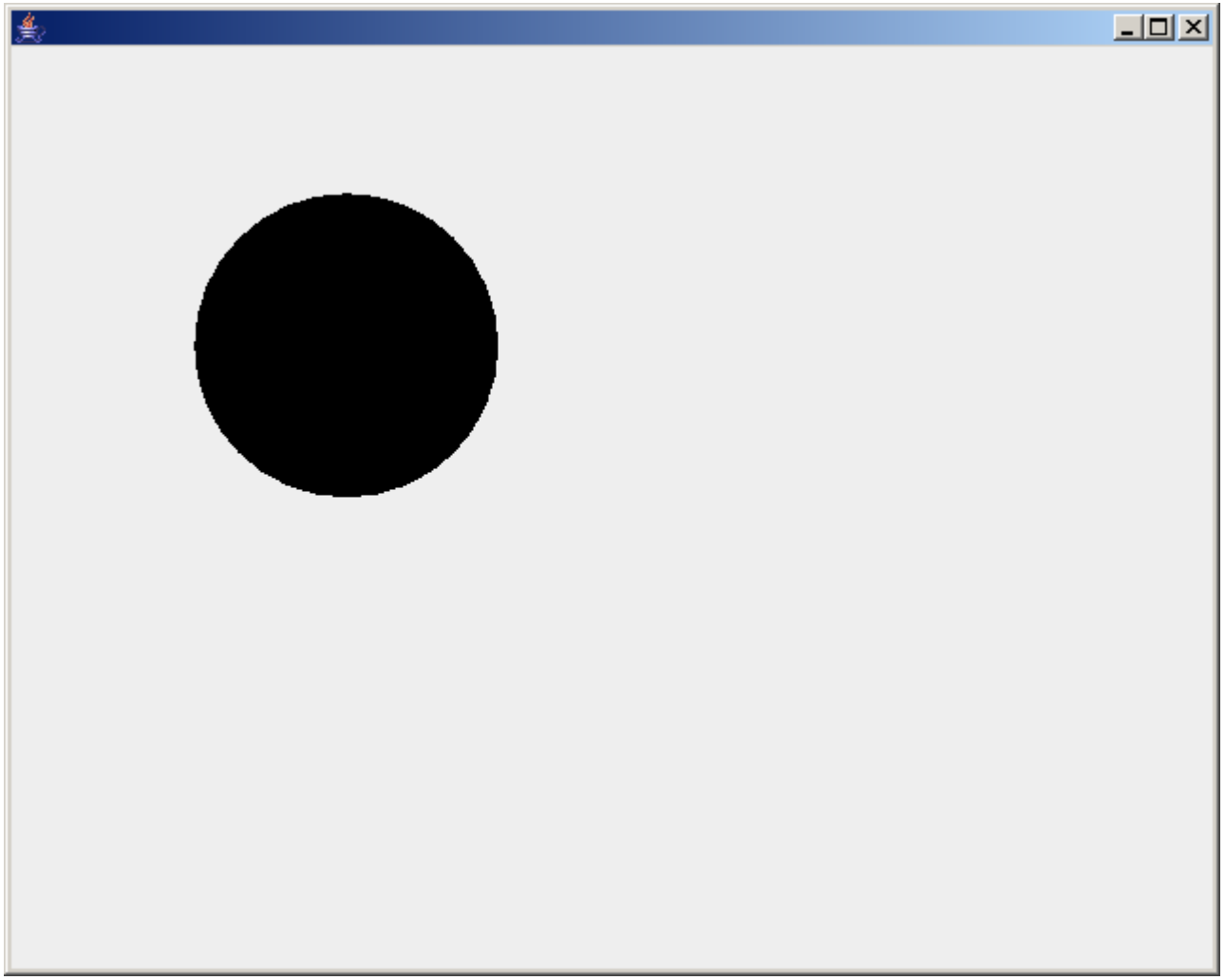
# Five.java



# Six.java



# Seven.java




# Rendering

- I Java2D finns en tydlig uppdelning mellan
  - Definition av objekt
  - Rendering, då objektet ritas genom att man skriver till pixels på utenheten
- Det finns möjligheter att styra hur objekt ritas. I den ”Grahics context” som används finns:
  - Vanliga attribut som färg och linjebredd
  - Rendering hints, t.ex. antialiasing


# Antialiasing

- Jämnar ut taggigheten hos lutande linjer och kurvor
- Ger i de flesta fall betydligt snyggare resultat
- Tar avsevärt längre tid att rendera bilderna

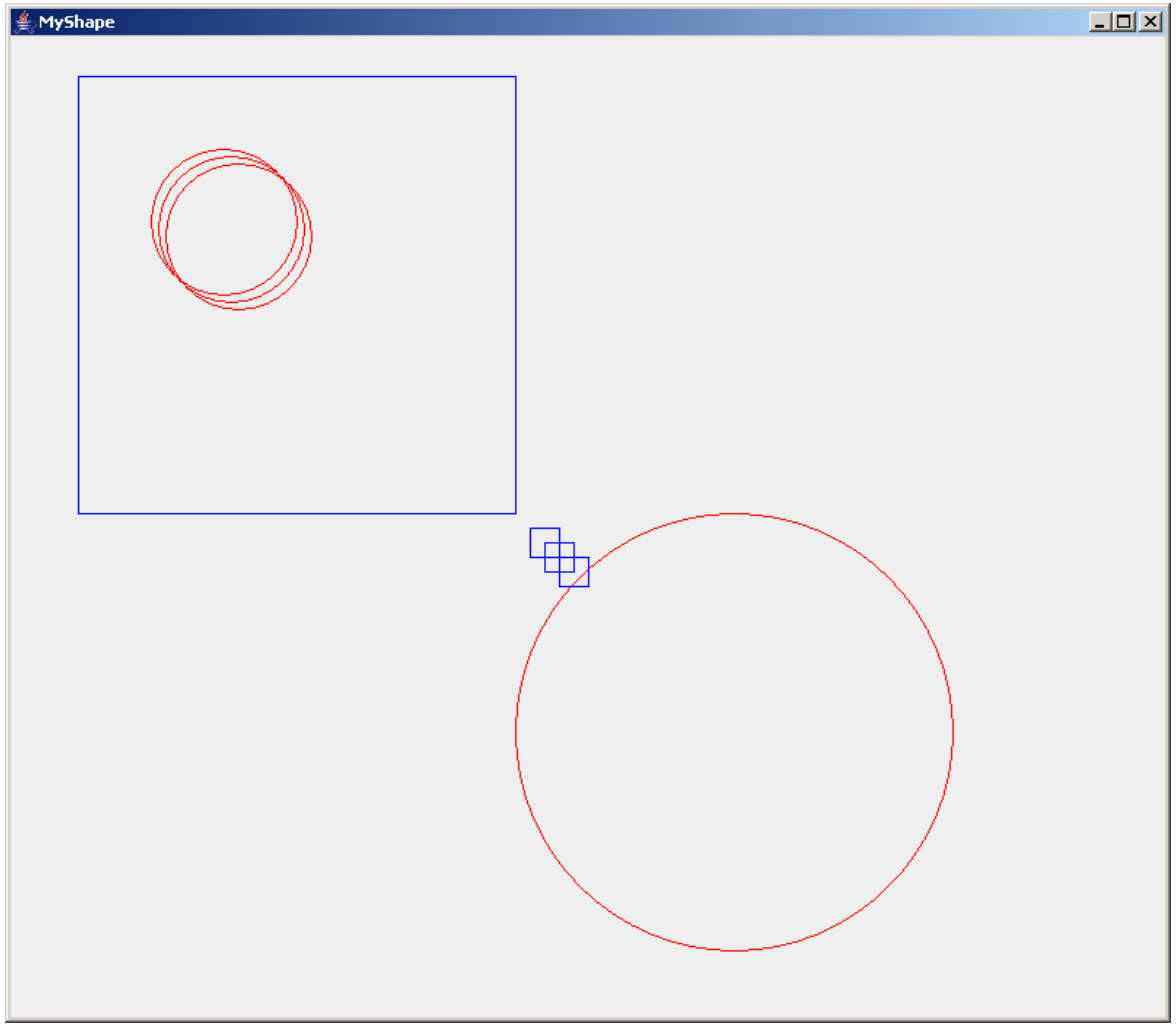

# java.awt.Shape

- Shape är ett interface
- Används för att beskriva alla objekt i Java2D
- Det som beskrivs är konturerna
- En kontur kan bestå av ett eller flera segment av räta linjer eller kurvor
- För vissa grundläggande geometriska former finns bekväma genvägar till att skapa färdiga slutna figurer utan att behöva specificera hela konturen segment för segment
- Konkreta implementationer av interfacet Shape finns i klasser i java.awt.geom.*

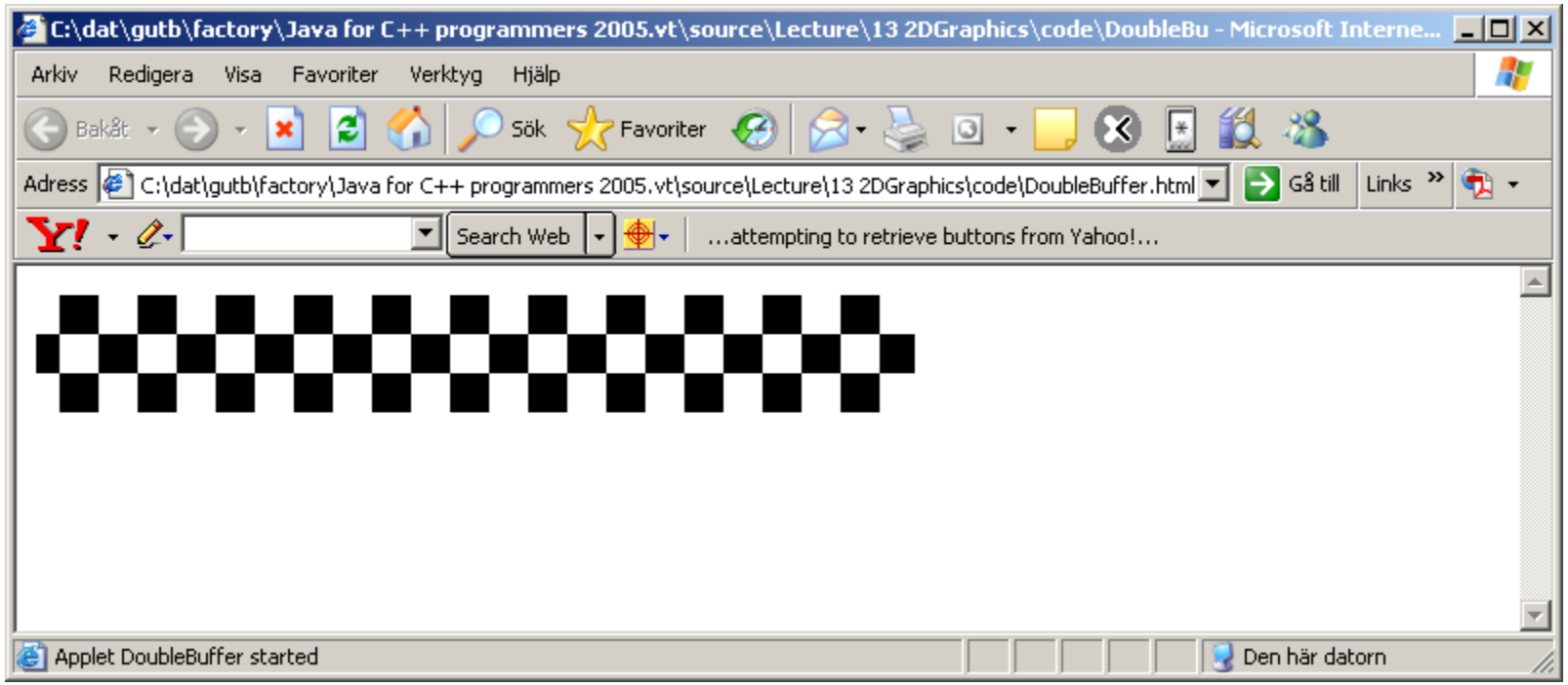

# Rectangle2D

- Klassen är abstrakt
- Det finns två konkreta implementationer
  - Flyttal enkel precision (float)
  - Flyttal dubbel precision (double)
- *public Rectangle2D.Double(double xpos, double ypos,double width, double height)*

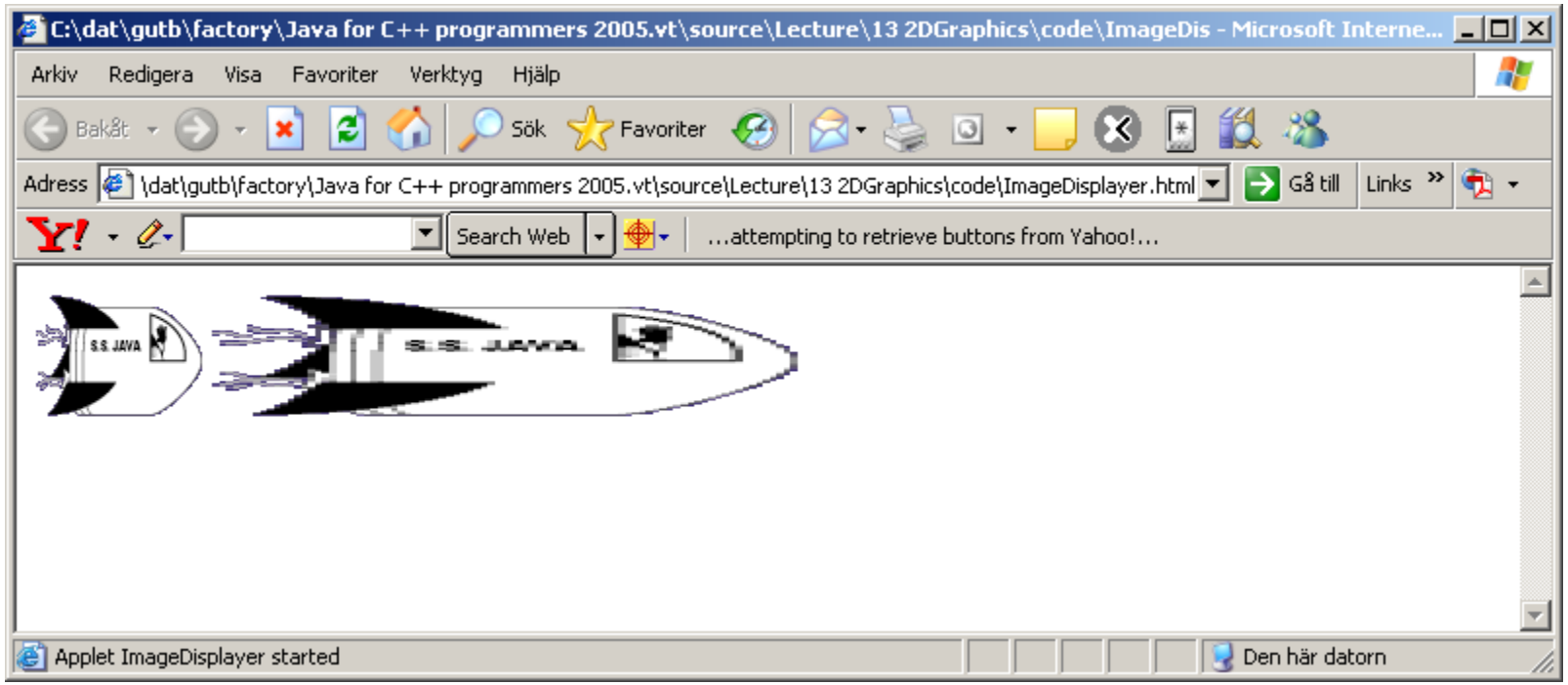
# MyShapeX.java



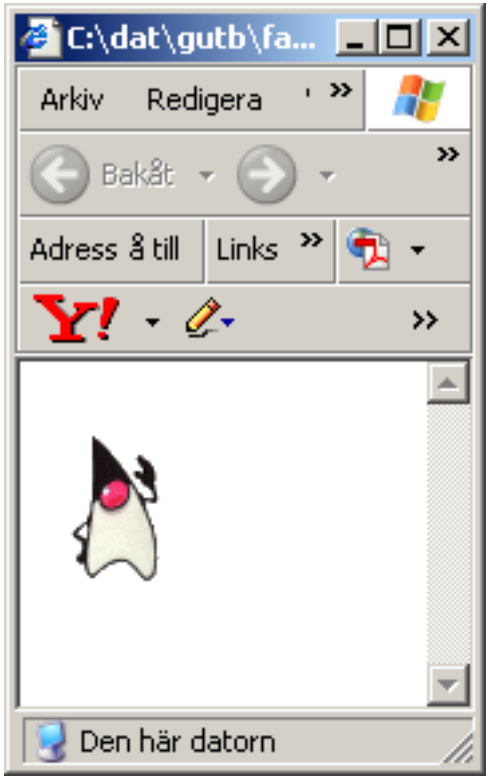
# DoubleBuffer.html



# ImageDisplayer.html



# ImageSequence.html



# RectangleDemo.html