

JAVA PROGRAMMING

Chapter 5

JAVA APPLET AND GRAPHICS PROGRAMMING

Detailed Study Notes for Engineering Students

With 15 Multiple Choice Questions (with Answer Key) and 10 Broad Questions

Contents of this Chapter

No.	Section	Topics covered
5.1	Applet Programming	Local and remote applets; how applets differ from applications; preparing to write applets; building applet code; applet life cycle; creating an executable applet; designing a web page; the APPLET tag; adding an applet to an HTML file; running the applet; passing parameters to an applet
5.2	Graphics Programming	The Graphics class; lines and rectangles; circles and ellipses; drawing arcs; drawing polygons; line graphs; using control loops in applets; drawing bar charts
	Chapter Practice	15 MCQs with answer key; 10 broad questions

5.1 Applet Programming

Java can be used to create two types of programs: **stand-alone applications** and **applets**. An applet is a small Java program that does not run on its own. It is embedded inside a web page (an HTML document) and is executed by a Java-enabled web browser or by a special testing tool called the appletviewer. Applets were introduced with the first release of Java and were largely responsible for its early popularity, because they turned static web pages into dynamic and interactive ones: animations, calculators, games and graphical charts could run directly inside a page.

Every applet is a class that extends the class `Applet` of the package `java.applet`. Because `Applet` itself inherits from `Panel`, `Container` and `Component` of the Abstract Window Toolkit (AWT), an applet automatically receives a rectangular display area on which text, shapes, images and user-interface components can be placed.

```
java.lang.Object
|
+-- java.awt.Component
|   |
|   +-- java.awt.Container
|       |
|       +-- java.awt.Panel
|           |
|           +-- java.applet.Applet
```

Main features of applets

- Applets are **embedded in HTML pages** and run by a browser or by the appletviewer.
- An applet has **no main() method**; the browser controls it by calling its life-cycle methods.
- Applets are **platform independent**: the bytecode (.class file) is downloaded and run by the Java Virtual Machine of the user's machine.
- Applets run in a **restricted environment (sandbox)** so that a downloaded program cannot harm the user's computer.
- Applets are **GUI based**: output is drawn on the applet window using the `Graphics` class.
- Applets are small, so they download quickly over a network.

Important note on the present status of applets

- The Applet API was deprecated in Java 9 and marked for removal in Java 17; major web browsers stopped supporting the Java plug-in years ago, and the appletviewer tool is not shipped with recent JDKs.
- Applets are still part of the engineering syllabus because they are an excellent way to learn the AWT, event-driven GUI programming and the `Graphics` class.
- To practise, use JDK 8 (which still includes appletviewer) or an IDE/tool that supports applets. The same drawing code can also be reused inside a `Frame` or `panel` of a normal application.

5.1.1 Local and Remote Applets

Applets can be classified according to *where they are stored* and *how they reach the user's computer*. On this basis there are two kinds: local applets and remote applets.

- **Local applet:** An applet that is developed locally and stored in the local system. When a web page containing it is opened, the browser searches for the .class file in the local file system and runs it. No Internet connection is required, so a local applet is fast to load.
- **Remote applet:** An applet that is developed by someone else and stored on a remote computer connected to the Internet. To load it, the browser must know its address (URL). The page specifies this address with the CODEBASE attribute of the APPLET tag. The class file is downloaded across the network and then executed on the user's machine.

Point	Local applet	Remote applet
Where stored	On the local computer	On a remote server on the Internet
Internet needed	No	Yes
How it is located	Using the CODE attribute; the file is searched in the local directory	Using CODEBASE (URL) together with CODE
Loading speed	Fast	Depends on network speed
Developed by	The user / local programmer	Another programmer or organisation
Security	Comparatively safe	Strictly restricted by the browser sandbox

Example of the tag used to load a remote applet:

```
<applet codebase="http://www.example.com/applets/"
        code="ClockApplet.class" width="200" height="100">
</applet>
```

What happens when a page containing an applet is opened?

1. The browser loads the HTML page and reads it from top to bottom.
2. On reaching the APPLET tag, it finds the name of the applet class (CODE) and its location (CODEBASE).
3. The class file (and any other classes it needs) is loaded - from the local disk or downloaded from the server.
4. The bytecode is checked by the bytecode verifier of the Java Virtual Machine.
5. The browser creates the applet object and calls its life-cycle methods `init()`, `start()` and `paint()` one after another.

5.1.2 How Applets Differ from Applications

An application is a stand-alone program that runs directly on the Java interpreter; an applet is a program that runs under the control of a browser. Because of this basic difference, the structure, execution and security rules of the two are quite different.

Feature	Applet	Application
Definition	Runs inside a web browser or appletviewer	Stand-alone program run directly by the Java interpreter
main() method	Not used	Required: <code>public static void main(String[] args)</code>
Starting point	Browser calls <code>init()</code> , <code>start()</code> , <code>paint()</code>	JVM calls <code>main()</code>
Base class	Must extend <code>Applet</code> (or <code>JApplet</code>)	No special base class needed

Feature	Applet	Application
How it is run	Embedded in HTML with the APPLET tag; run by browser/appletviewer	java ClassName
Compilation	javac plus an HTML file is required	javac only
Output	Drawn on the applet window using Graphics (paint method)	Console (System.out) or GUI frames
Security	Restricted; cannot access local files or run programs	Full access to system resources
Network access	Only to the host from which it came	Unrestricted
Life cycle	init, start, stop, destroy managed by the browser	Runs until main() ends or System.exit() is called
Use	Small interactive web-based programs	General-purpose software

Security restrictions on applets

An applet downloaded from the Internet is untrusted, so the browser runs it in a "sandbox" with the following restrictions:

- It cannot read or write files on the user's computer.
- It cannot open a network connection to any computer except the host from which it was downloaded.
- It cannot run any program on the user's computer.
- It cannot load libraries or define native methods.
- It cannot read most system properties.
- Any window it opens carries a warning banner so that the user knows it is an applet window.

5.1.3 Preparing to Write Applets

An applet is not started by the programmer's own main() but by the browser. Therefore, before writing an applet, certain requirements must be fulfilled and certain rules must be understood.

- Install the **Java Development Kit (JDK)**, which provides the compiler javac and (in older versions) the appletviewer, and have a Java-enabled browser available.
- Learn the **basics of HTML**, because an applet is placed inside a web page.
- Learn the **AWT classes** used by applets: Applet, Graphics, Color, Font, Polygon.
- Decide what the applet will do, the **size of its window** (width and height) and whether **parameters** will be passed from the web page.
- Remember that an applet does **not use console input** (System.in); input is taken through GUI components, mouse and keyboard events or parameters.
- Decide which of the life-cycle methods (init, start, paint, stop, destroy) must be overridden.

Package	Used for
java.applet.*	The Applet class, AppletContext, AudioClip
java.awt.*	Graphics, Color, Font, Polygon, Dimension and GUI components such as Button, Label, TextField
java.awt.event.*	Event handling (needed only when the applet responds to mouse, keyboard or button events)

Steps in developing and running an applet

1. **Building the applet code** - write the source file (.java).
2. **Creating an executable applet** - compile it to a .class file.
3. **Designing a web page** - write an HTML file containing the APPLET tag.
4. **Running the applet** - open the HTML file with the appletviewer or a browser.

5.1.4 Building Applet Code

The applet code is a Java class which extends Applet. The general skeleton of an applet is shown below. Only the methods that are required need to be written, because the Applet class already supplies empty versions of the others.

```
import java.applet.Applet;
import java.awt.Graphics;

public class AppletClassName extends Applet {
    public void init()    { /* initialisation code    */ }
    public void start()   { /* start or resume work   */ }
    public void paint(Graphics g) { /* drawing code   */ }
    public void stop()    { /* suspend work          */ }
    public void destroy() { /* final clean-up        */ }
}
```

Rules for writing applet code

- The class must be declared **public**, because the browser (code outside the program) has to access it.
- The class must **extend java.applet.Applet**.
- The source file name must be exactly the same as the class name, e.g. HelloApplet.java.
- All drawing and text output is done in paint(Graphics g). The Graphics object g is supplied automatically by the system.
- The import statements for java.applet.Applet and java.awt.Graphics (or java.awt.*) must be present.

Example 1 - A simple applet

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("Hello Java Applet", 20, 30);
    }
}
```

Statement	Meaning
import java.applet.Applet;	Makes the Applet class available to the program.
import java.awt.Graphics;	Makes the Graphics class (used for drawing) available.
public class HelloApplet extends Applet	Declares a public class that inherits all the features of Applet.
public void paint(Graphics g)	Called automatically whenever the applet display has to be drawn; g is the graphics context.

Statement	Meaning
<code>g.drawString("Hello Java Applet", 20, 30);</code>	Draws the text with its starting point (baseline) at x = 20, y = 30 pixels from the top-left corner.

Output: The applet window displays the text **Hello Java Applet** near its top-left corner.

Useful method

- `showStatus(String msg)` displays a message in the status bar of the browser or appletviewer, for example `showStatus("Applet is running");`.

5.1.5 Applet Life Cycle

Every applet passes through a series of stages from the moment it is loaded until it is removed from memory. These stages are controlled by the browser, which calls specific methods of the applet at the right times. The methods `init()`, `start()`, `stop()` and `destroy()` belong to the `Applet` class, while `paint()` is inherited from `Component` and is called whenever the display must be drawn. Together they form the applet life cycle.

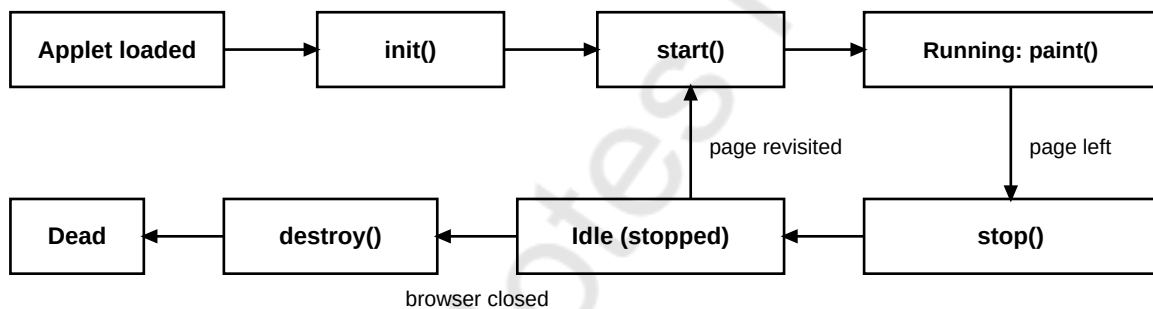


Figure 5.1 Applet life cycle

Method	When it is called	Typical use	Calls
<code>init()</code>	When the applet is loaded for the first time, before any other method	Initialise variables, create objects, read parameters, load images and fonts, set colours	Once
<code>start()</code>	Immediately after <code>init()</code> , and again every time the user returns to the page	Start or resume threads, animation and sound	Many times
<code>paint(Graphics g)</code>	After <code>start()</code> , whenever the window is uncovered or resized, or when <code>repaint()</code> is called	Draw text, shapes and images	Many times
<code>stop()</code>	When the user leaves the page or minimises the browser window	Suspend threads, stop animation and sound to save CPU time	Many times
<code>destroy()</code>	When the applet is about to be removed from memory (browser or page closed)	Release resources, close files and connections, final clean-up	Once

States of an applet

- **Born (initialisation) state:** the applet has been loaded and `init()` executes. It happens only once.

- **Running state:** `start()` has been called and the applet is active and displays its output through `paint()`. The applet re-enters this state each time the user comes back to the page.
- **Idle (stopped) state:** `stop()` has been called because the user left the page. The applet is still in memory and can return to the running state through `start()`.
- **Dead state:** `destroy()` has been called and the applet is removed from memory. It cannot be restarted without reloading the page.

repaint(), update() and paint()

The programmer should never call `paint()` directly. To redraw the applet, call `repaint()`. This asks the AWT to call `update()`, which clears the applet area to the background colour and then calls `paint()`.

Event	Methods called, in order
Applet is loaded for the first time	<code>init()</code> -> <code>start()</code> -> <code>paint()</code>
User moves to another page	<code>stop()</code>
User returns to the page	<code>start()</code> -> <code>paint()</code>
Applet window is covered and then uncovered	<code>paint()</code>
Program calls <code>repaint()</code>	<code>update()</code> -> <code>paint()</code>
Browser or appletviewer is closed	<code>stop()</code> -> <code>destroy()</code>

Example 2 - Demonstrating the life cycle

```
import java.applet.Applet;
import java.awt.Graphics;

public class LifeCycleDemo extends Applet {
    String msg = "";

    public void init()    { msg += "init() "; }
    public void start()   { msg += "start() "; }

    public void paint(Graphics g) {
        msg += "paint() ";
        g.drawString(msg, 10, 20);
    }

    public void stop()    { System.out.println("stop() called"); }
    public void destroy() { System.out.println("destroy() called"); }
}
```

Output: The window first shows **init() start() paint()**. Each time the window is redrawn, another "paint()" is appended to the message. The messages of `stop()` and `destroy()` appear on the Java console, not in the applet window.

5.1.6 Creating an Executable Applet

The source file is not executable. It must be translated by the Java compiler into **bytecode**; the resulting **.class** file is the executable applet that the browser loads. The steps are:

1. Save the program as `HelloApplet.java` (file name = name of the public class).
2. Open a command prompt in the folder that contains the file.
3. Compile the program with the command shown below.
4. If there are no errors, the file `HelloApplet.class` is created in the same folder.

- Keep the class file where the HTML page can find it - normally in the same directory as the HTML file (otherwise give its location with CODEBASE).

```
javac HelloApplet.java
```

Problem	Cause	Remedy
class HelloApplet is public, should be declared in a file named HelloApplet.java	File name differs from the class name	Rename the file to match the class name exactly
cannot find symbol: class Applet	import statement is missing	Add import java.applet.Applet;
Applet not loaded / ClassNotFoundException when running	The .class file is not in the directory given by CODE or CODEBASE	Place the class file in the correct directory or correct the tag
Blank applet window	paint() not written, or drawing is outside the window, or WIDTH/HEIGHT too small	Check the drawing code and the size attributes

5.1.7 Designing a Web Page

A web page is written in HTML (HyperText Markup Language). HTML uses **tags** enclosed in angle brackets to describe the structure of the page. A page has a **head** section, which holds information such as the title, and a **body** section, which holds everything that is displayed. The applet is placed in the body with the APPLET tag.

Tag	Purpose
<html> ... </html>	Root tag; encloses the entire document
<head> ... </head>	Contains information about the page such as its title
<title> ... </title>	Title shown in the title bar of the browser
<body> ... </body>	Contains the visible content of the page
<h1> ... </h1> to <h6>	Headings of decreasing size
<p>	Starts a new paragraph
, <i>	Bold and italic text
 	Line break
<applet> ... </applet>	Embeds an applet in the page

```
<html>
<head>
<title>My First Applet Page</title>
</head>
<body>
<h1>Welcome to Applet Programming</h1>
<p>The applet will be displayed below.</p>
</body>
</html>
```

While designing a page for an applet, give it a meaningful title, add a short description, reserve a display area of suitable width and height for the applet, and provide alternate text for browsers that cannot run Java.

5.1.8 The APPLET Tag

The APPLET tag is used to place an applet in an HTML page. It tells the browser *which* applet to load, *where* to find it and *how much space* to give it. The general form is:

```
<APPLET
  CODE      = "AppletClassName.class"
  WIDTH     = pixels
  HEIGHT    = pixels
  [CODEBASE = "url_or_directory"]
  [ALT      = "alternate text"]
  [NAME     = "applet_name"]
  [ALIGN    = alignment]
  [VSPACE   = pixels] [HSPACE = pixels]
  [ARCHIVE  = "file.jar"] >

  [<PARAM NAME = "name" VALUE = "value">]
  [alternate HTML text]
</APPLET>
```

Attribute	Meaning	Required?
CODE	Name of the compiled applet class file, e.g. HelloApplet.class	Yes
WIDTH	Width of the applet display area in pixels	Yes
HEIGHT	Height of the applet display area in pixels	Yes
CODEBASE	URL or directory where the class file is stored; by default, the directory of the HTML file	No
ALT	Text displayed if the browser understands the tag but cannot run applets	No
NAME	A name for the applet so that other applets on the page can communicate with it	No
ALIGN	Alignment of the applet area (LEFT, RIGHT, TOP, MIDDLE, BOTTOM)	No
VSPACE, HSPACE	Vertical and horizontal space (in pixels) around the applet	No
ARCHIVE	A JAR file holding the applet classes, so that all are downloaded in one request	No
PARAM	A child tag with NAME and VALUE used to pass parameters to the applet	No

- Tag and attribute names are not case sensitive.
- Text written between <APPLET> and </APPLET> is displayed by browsers that do not support applets.
- The APPLET tag was deprecated in HTML 4 in favour of the OBJECT tag and is obsolete in HTML5, but it is the form used in the syllabus and by the appletviewer.

5.1.9 Adding Applet to HTML File

1. Open a text editor and create a new HTML file (for example HelloApplet.html).
2. Write the basic HTML structure: html, head, title and body tags.
3. Inside the body, add the APPLET tag with CODE, WIDTH and HEIGHT.
4. Optionally add PARAM tags and alternate text between the opening and closing APPLET tags.
5. Save the file in the same folder as the .class file.

```
<html>
<head>
```

```
<title>Hello Applet Page</title>
</head>
<body>
<h2>My First Applet</h2>
<applet code="HelloApplet.class" width="300" height="100">
Your browser does not support Java applets.
</applet>
</body>
</html>
```

Alternative method. For quick testing, the APPLET tag can be written as a comment at the top of the Java source file. The appletviewer reads this tag, so a separate HTML file is not needed:

```
/*
<applet code="HelloApplet.class" width="300" height="100">
</applet>
*/
import java.applet.Applet;
import java.awt.Graphics;

public class HelloApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("Hello Java Applet", 20, 30);
    }
}
```

This is run with `appletviewer HelloApplet.java`.

5.1.10 Running the Applet

After compiling the applet and preparing the HTML file, the applet can be run in two ways:

1. **Using the appletviewer** - a tool of the JDK that displays only the applet and ignores the other HTML content.
2. **Using a Java-enabled web browser** - the HTML file is opened in the browser, which displays the full page with the applet embedded in it.

```
javac HelloApplet.java
appletviewer HelloApplet.html
```

Point	appletviewer	Web browser
Needs	JDK (older versions)	A Java plug-in / Java-enabled browser
Displays	Only the applet window	The whole HTML page including the applet
Command / action	<code>appletviewer file.html</code>	Open the HTML file in the browser
Best for	Quick testing and debugging	Final view as the user will see it

Error messages and the output of `System.out.println()` inside an applet appear on the **Java console**, which is helpful for debugging. If the applet does not appear, check that the `.class` file is in the right folder, that the CODE name is spelt correctly (including capital letters) and that WIDTH and HEIGHT are not zero.

5.1.11 Passing Parameters to Applet

Just as command-line arguments let us pass values to the `main()` method of an application, **parameters** let us pass values from the HTML page to an applet. This allows the same applet to behave differently on different pages without changing or recompiling its code.

Parameters are given with the <PARAM> tag placed between <APPLET> and </APPLET>, each with a NAME and a VALUE. The applet reads a parameter with the method:

```
String getParameter(String name)
```

1. Add one <PARAM NAME="..." VALUE="..."> tag for each value inside the APPLET tag.
2. In the applet, normally in init(), call getParameter("name") to read the value.
3. The method always returns a **String**, and returns **null** if the parameter is not found. Check for null and supply a default value.
4. Convert the string to a number with Integer.parseInt() or Double.parseDouble() when needed.

Example 3 - Applet that receives parameters

HTML file:

```
<html>
<body>
<applet code="ParamApplet.class" width="300" height="120">
  <param name="name" value="Ravi">
  <param name="age" value="19">
  <param name="course" value="Computer Engineering">
</applet>
</body>
</html>
```

Java file:

```
import java.applet.Applet;
import java.awt.Graphics;

public class ParamApplet extends Applet {
    String name, course;
    int age;

    public void init() {
        name = getParameter("name");
        course = getParameter("course");
        String a = getParameter("age");

        if (name == null) name = "Guest";
        if (course == null) course = "Not given";
        try {
            age = Integer.parseInt(a);
        } catch (Exception e) {
            age = 0;
        }
    }

    public void paint(Graphics g) {
        g.drawString("Name : " + name, 20, 30);
        g.drawString("Age : " + age, 20, 55);
        g.drawString("Course : " + course, 20, 80);
    }
}
```

PARAM NAME	VALUE (String)	Variable in the applet
name	Ravi	name (String)
age	19	age (int, after Integer.parseInt)
course	Computer Engineering	course (String)

Output: The window displays the three lines Name : Ravi, Age : 19 and Course : Computer Engineering. If a PARAM tag is removed, the default value in the code is displayed instead.

Quick revision - 5.1 Applet Programming

- Applet = Java program embedded in HTML, extends `java.applet.Applet`, has no `main()`.
- Life cycle: `init()` (once) -> `start()` -> `paint()` -> `stop()` -> `destroy()` (once).
- Steps: write .java -> compile with `javac` -> write HTML with APPLET tag -> run with `appletviewer` or browser.
- APPLET tag needs CODE, WIDTH and HEIGHT; use CODEBASE for remote applets; use PARAM for parameters.
- `getParameter()` returns a String (or null).
- Untrusted applets cannot access local files, run programs or connect to hosts other than their own.

5.2 Graphics Programming

Applets are mainly used to produce graphical output. Java provides drawing facilities through the `Graphics` class of the `java.awt` package. Inside an applet the system passes a `Graphics` object to the `paint()` method, and calling the methods of this object draws text and shapes on the applet window. This section explains the drawing methods for lines, rectangles, ovals, arcs and polygons, and then uses them, with loops, to draw line graphs and bar charts.

5.2.1 The Graphics Class

The `Graphics` class is an **abstract class** that represents the **graphics context**, that is, the drawing surface together with its current colour, font and other settings. A programmer does not create a `Graphics` object with `new`; it is received as the parameter of `paint(Graphics g)`.

Coordinate system

- The origin **(0, 0)** is at the **top-left corner** of the applet window.
- The x-coordinate increases **to the right** and the y-coordinate increases **downwards**.
- All measurements are in **pixels**. The size of the window is set by `WIDTH` and `HEIGHT` in the `APPLET` tag.

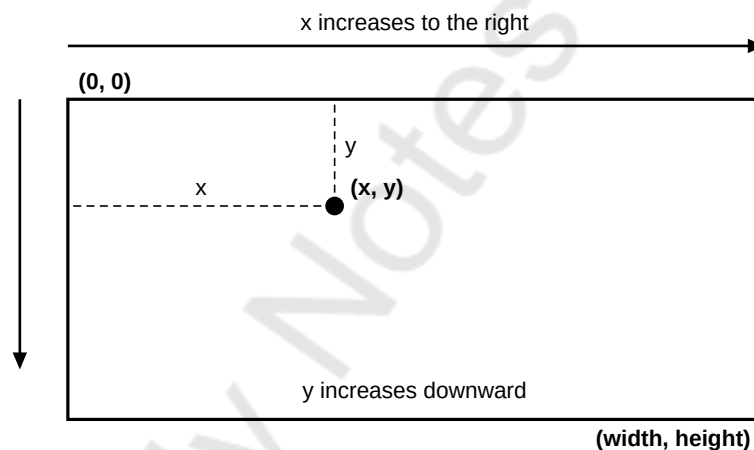


Figure 5.2 Coordinate system of the applet window

Commonly used methods of the Graphics class

Method	Purpose
<code>drawString(String s, int x, int y)</code>	Draws text starting at (x, y)
<code>drawLine(int x1, int y1, int x2, int y2)</code>	Draws a straight line between two points
<code>drawRect(x, y, w, h) / fillRect(x, y, w, h)</code>	Draws an outline / a solid rectangle
<code>clearRect(x, y, w, h)</code>	Erases a rectangular area (fills it with the background colour)
<code>drawRoundRect(x, y, w, h, aw, ah) / fillRoundRect(...)</code>	Rectangle with rounded corners
<code>draw3DRect(x, y, w, h, raised) / fill3DRect(...)</code>	Rectangle with a 3-D (raised or sunken) effect
<code>drawOval(x, y, w, h) / fillOval(x, y, w, h)</code>	Draws an outline / a solid oval (ellipse or circle)
<code>drawArc(x, y, w, h, start, sweep) / fillArc(...)</code>	Draws an arc / a filled pie slice
<code>drawPolygon(int[] xs, int[] ys, int n) / fillPolygon(...)</code>	Draws an outline / a solid polygon
<code>drawPolyline(int[] xs, int[] ys, int n)</code>	Draws connected line segments without closing the figure
<code>setColor(Color c) / getColor()</code>	Sets / returns the current drawing colour
<code>setFont(Font f) / getFont()</code>	Sets / returns the current font

As a rule, methods whose names begin with **draw** produce an **outline** and methods beginning with **fill** produce a **solid** shape in the current colour.

Colours and fonts

The Color class provides predefined constants such as `Color.black`, `white`, `red`, `green`, `blue`, `yellow`, `cyan`, `magenta`, `orange`, `pink`, `gray`. A custom colour is created with `new Color(r, g, b)`, where each value lies between 0 and 255. A font is created with `new Font(name, style, size)`, where the style is `Font.PLAIN`, `Font.BOLD` or `Font.ITALIC`. The methods `setBackground(Color)` and `setForeground(Color)` of the applet set the background and default drawing colours.

```
import java.applet.Applet;
import java.awt.*;

public class TextDemo extends Applet {
    public void paint(Graphics g) {
        g.setColor(Color.blue);
        g.setFont(new Font("Serif", Font.BOLD, 20));
        g.drawString("Graphics in Java", 30, 40);

        g.setColor(new Color(200, 0, 100)); // custom colour
        g.setFont(new Font("SansSerif", Font.ITALIC, 16));
        g.drawString("Custom colour text", 30, 70);
    }
}
```

5.2.2 Lines and Rectangles

Drawing a line

```
void drawLine(int x1, int y1, int x2, int y2)
```

It draws a straight line from the point (x1, y1) to the point (x2, y2) in the current colour. If y1 = y2 the line is horizontal; if x1 = x2 it is vertical; and drawLine(x, y, x, y) draws a single point.

Drawing rectangles

Method	Description
drawRect(x, y, w, h)	Outline of a rectangle whose top-left corner is (x, y), with width w and height h
fillRect(x, y, w, h)	Solid rectangle filled with the current colour
clearRect(x, y, w, h)	Fills the rectangle with the background colour, i.e. erases that area
drawRoundRect(x, y, w, h, arcWidth, arcHeight)	Rectangle with rounded corners; arcWidth and arcHeight give the horizontal and vertical diameter of the corner arcs
fillRoundRect(x, y, w, h, arcWidth, arcHeight)	Solid rounded rectangle
draw3DRect(x, y, w, h, raised)	Outline with a 3-D effect; raised = true gives a raised look, false a sunken look
fill3DRect(x, y, w, h, raised)	Solid rectangle with a 3-D effect

- The last two numbers of drawRect are the **width and height**, not the coordinates of the opposite corner.
- A rectangle with **w = h** is a square.
- Use setColor() before drawing to choose the colour of the shape.

Example 4 - Lines and rectangles

```
import java.applet.Applet;
import java.awt.*;

public class LineRectDemo extends Applet {
    public void paint(Graphics g) {
        g.drawLine(20, 20, 220, 20);           // horizontal line
        g.drawLine(20, 30, 220, 80);           // slanting line
        g.drawRect(20, 100, 120, 60);           // hollow rectangle
        g.fillRect(160, 100, 60, 60);           // solid square
        g.drawRoundRect(20, 180, 120, 60, 25, 25); // rounded rectangle
        g.fill3DRect(160, 180, 60, 60, true);   // raised 3-D rectangle
    }
}
```

Output: Two lines at the top, a hollow rectangle and a solid square in the middle row, and a rounded rectangle and a 3-D rectangle in the bottom row.

5.2.3 Circle and Ellipse

Java has no separate method for a circle. Both circles and ellipses are drawn with drawOval() (outline) and fillOval() (solid):

```
void drawOval(int x, int y, int width, int height)
void fillOval(int x, int y, int width, int height)
```

The oval is drawn **inside an imaginary bounding rectangle** whose top-left corner is (x, y) and whose size is width x height. When **width = height** the oval is a **circle**; otherwise it is an **ellipse**.

Drawing a circle from its centre and radius

- For a circle with centre (cx, cy) and radius r use: `drawOval(cx - r, cy - r, 2 * r, 2 * r)`
- Worked example: centre (100, 100), radius 40 -> `g.drawOval(60, 60, 80, 80);`

Figure	Statement	Remark
Hollow circle	<code>g.drawOval(60, 60, 80, 80);</code>	width = height = 80
Solid circle	<code>g.fillOval(60, 60, 80, 80);</code>	filled with current colour
Wide ellipse	<code>g.drawOval(20, 20, 150, 60);</code>	width greater than height
Tall ellipse	<code>g.fillOval(200, 20, 60, 120);</code>	height greater than width

Example 5 - Circles and ellipses

```
import java.applet.Applet;
import java.awt.*;

public class OvalDemo extends Applet {
    public void paint(Graphics g) {
        g.drawOval(20, 20, 100, 100);    // circle
        g.fillOval(140, 20, 100, 100);  // solid circle
        g.drawOval(20, 140, 160, 70);   // horizontal ellipse
        g.fillOval(200, 140, 60, 100);  // vertical ellipse

        int cx = 150, cy = 120, r = 50; // circle using centre and radius
        g.drawOval(cx - r, cy - r, 2 * r, 2 * r);
    }
}
```

5.2.4 Drawing Arcs

An arc is a part of the outline of an ellipse. It is drawn with:

```
void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)
void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)
```

- **x, y, width, height** describe the bounding rectangle of the complete oval from which the arc is cut.
- **startAngle** is the angle, in degrees, at which the arc begins. The angle 0 is at the **3 o'clock** position.
- **arcAngle** is the angular extent (sweep) of the arc. A **positive** value sweeps **anticlockwise**; a **negative** value sweeps **clockwise**.
- **fillArc()** draws a solid sector (pie slice) by joining the ends of the arc to the centre.

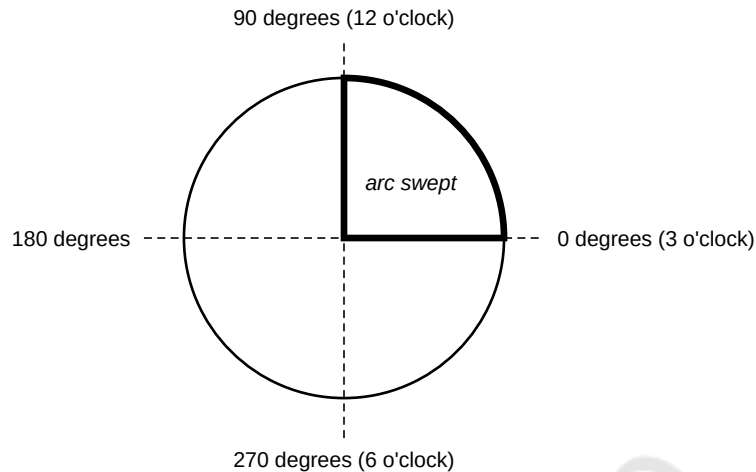


Figure 5.3 Angle convention for arcs (the thick portion is drawArc(..., 0, 90))

Call (bounding box 100 x 100)	Result
<code>drawArc(20, 20, 100, 100, 0, 90)</code>	Quarter circle from 3 o'clock to 12 o'clock (anticlockwise)
<code>drawArc(20, 20, 100, 100, 0, 180)</code>	Upper semicircle
<code>drawArc(20, 20, 100, 100, 180, 180)</code>	Lower semicircle (from 9 o'clock through 6 o'clock to 3 o'clock)
<code>drawArc(20, 20, 100, 100, 90, -90)</code>	Quarter circle from 12 o'clock to 3 o'clock, drawn clockwise
<code>drawArc(20, 20, 100, 100, 0, 360)</code>	A complete circle
<code>fillArc(20, 20, 100, 100, 0, 270)</code>	Solid three-quarter sector

Example 6 - Arcs, a pie chart and a smiley face

```
import java.applet.Applet;
import java.awt.*;

public class ArcDemo extends Applet {
    public void paint(Graphics g) {
        g.drawArc(20, 20, 100, 100, 0, 90);        // quarter arc
        g.drawArc(140, 20, 100, 100, 0, 180);      // upper semicircle
        g.fillArc(260, 20, 100, 100, 0, 270);      // three-quarter sector

        // pie chart: each start angle = previous start + previous sweep
        g.setColor(Color.red);   g.fillArc(20, 150, 150, 150, 0, 120);
        g.setColor(Color.green); g.fillArc(20, 150, 150, 150, 120, 140);
        g.setColor(Color.blue);  g.fillArc(20, 150, 150, 150, 260, 100);

        // smiley face
        g.setColor(Color.black);
        g.drawOval(220, 150, 130, 130);            // face
        g.fillOval(255, 190, 12, 12);              // left eye
        g.fillOval(305, 190, 12, 12);              // right eye
        g.drawArc(245, 205, 80, 55, 180, 180);     // smile (lower half of an oval)
    }
}
```

5.2.5 Drawing Polygons

A polygon is a closed figure bounded by straight lines. In Java a polygon is described by the coordinates of its corners, which are stored in two integer arrays - one for the x-coordinates and one for the y-coordinates.

```
void drawPolygon(int xPoints[], int yPoints[], int nPoints)
void fillPolygon(int xPoints[], int yPoints[], int nPoints)
void drawPolygon(Polygon p)
void fillPolygon(Polygon p)
void drawPolyline(int xPoints[], int yPoints[], int nPoints)
```

- **xPoints, yPoints** hold the x and y values of the corners; the i-th corner is (xPoints[i], yPoints[i]).
- **nPoints** is the number of corners (a triangle has 3, a pentagon 5).
- drawPolygon() automatically **closes** the figure by joining the last point to the first; drawPolyline() does **not** close it.
- The **Polygon** class can also be used: create it with new Polygon(), add corners with addPoint(x, y) and pass it to drawPolygon(p) or fillPolygon(p).

Example 7 - Polygons

```
import java.applet.Applet;
import java.awt.*;

public class PolygonDemo extends Applet {
    public void paint(Graphics g) {
        // triangle using arrays
        int tx[] = {60, 20, 100};
        int ty[] = {20, 100, 100};
        g.drawPolygon(tx, ty, 3);

        // filled pentagon using arrays
        int px[] = {200, 250, 230, 170, 150};
        int py[] = {20, 60, 120, 120, 60};
        g.fillPolygon(px, py, 5);

        // polygon using the Polygon class
        Polygon p = new Polygon();
        p.addPoint(60, 150);
        p.addPoint(110, 150);
        p.addPoint(130, 200);
        p.addPoint(80, 230);
        p.addPoint(30, 200);
        g.drawPolygon(p);
    }
}
```

5.2.6 Line Graphs

A line graph shows how one quantity (for example sales or temperature) changes with another (for example year or time). Points are plotted for the data values and consecutive points are joined by straight lines. It is drawn with drawLine() for the axes and the joining lines, fillOval() for the point markers and drawString() for the labels.

Steps for drawing a line graph

1. Draw the x-axis and y-axis with drawLine().
2. Choose the origin of the graph, the horizontal spacing between points (stepX) and a scale (pixels per unit of data).
3. Convert each data value to screen coordinates (see the formula below).

4. Plot each point and join it to the previous point with `drawLine()`.
5. Write the labels of the axes and the points with `drawString()`.

Conversion formula

- $x = \text{originX} + (i + 1) * \text{stepX}$
- $y = \text{originY} - \text{value} * \text{scale}$
- The subtraction is needed because the y-coordinate of the screen increases downwards, while a graph value increases upwards.

Example 8 - Line graph of yearly sales

```
import java.applet.Applet;
import java.awt.*;

public class LineGraphApplet extends Applet {
    int sales[] = {40, 70, 55, 90, 120, 100};

    public void paint(Graphics g) {
        int originX = 40, originY = 220;
        int stepX = 50; // distance between points

        g.drawLine(originX, originY, originX, 20); // y-axis
        g.drawLine(originX, originY, originX + 330, originY); // x-axis
        g.drawString("Sales", 5, 15);
        g.drawString("Year", originX + 335, originY + 4);

        for (int i = 0; i < sales.length; i++) {
            int x = originX + (i + 1) * stepX;
            int y = originY - sales[i]; // scale = 1 pixel per unit
            g.fillOval(x - 3, y - 3, 6, 6); // point marker
            if (i > 0) {
                int px = originX + i * stepX; // previous point
                int py = originY - sales[i - 1];
                g.drawLine(px, py, x, y); // join to previous point
            }
            g.drawString("" + (2020 + i), x - 12, originY + 15);
        }
    }
}
```

i	Year	Value	$x = 40 + (i+1) \times 50$	$y = 220 - \text{value}$
0	2020	40	90	180
1	2021	70	140	150
2	2022	55	190	165
3	2023	90	240	130
4	2024	120	290	100
5	2025	100	340	120

Suitable APPLET tag: `<applet code="LineGraphApplet.class" width="400" height="260">`.

5.2.7 Using Control Loops in Applets

Many drawings consist of the same shape repeated at regular positions. Instead of writing many nearly identical statements, we place the drawing statement inside a **control loop** (for, while or do-while) and use the **loop variable in the coordinates**. A **nested loop** (a loop inside a loop) is used for two-dimensional patterns such as grids. Loops are also used to go through the arrays of data in line graphs and bar charts.

Pattern	Loop idea
Concentric circles or rectangles	A for loop increases the size by a fixed amount and adjusts x and y to keep the centre
Row of shapes	The loop variable multiplies the spacing in the x-coordinate
Grid or chessboard	Nested loops: the outer loop for rows, the inner loop for columns
Scale marks on an axis	A loop draws short lines at equal intervals
Graphs and charts	A loop reads each element of the data array and draws its point or bar

Example 9 - Concentric circles

```
import java.applet.Applet;
import java.awt.*;

public class CircleLoopDemo extends Applet {
    public void paint(Graphics g) {
        int cx = 150, cy = 150;
        for (int r = 10; r <= 100; r += 10) {
            g.drawOval(cx - r, cy - r, 2 * r, 2 * r);
        }
    }
}
```

The loop draws ten circles with radii 10, 20, ... 100, all having the same centre (150, 150).

Example 10 - Chessboard using nested loops

```
import java.applet.Applet;
import java.awt.*;

public class ChessBoard extends Applet {
    public void paint(Graphics g) {
        int size = 30; // side of one square
        for (int row = 0; row < 8; row++) {
            for (int col = 0; col < 8; col++) {
                int x = 20 + col * size;
                int y = 20 + row * size;
                if ((row + col) % 2 == 0)
                    g.fillRect(x, y, size, size); // dark square
                else
                    g.drawRect(x, y, size, size); // light square
            }
        }
    }
}
```

The outer loop runs through the 8 rows and the inner loop through the 8 columns, so the body executes 64 times. The condition $(row + col) \% 2 == 0$ alternates the squares. A suitable applet size is 300 x 300.

Scale marks on an axis can be drawn with a single loop:

```
for (int i = 0; i <= 10; i++) {
    g.drawLine(40 + i * 30, 215, 40 + i * 30, 225); // a tick every 30 pixels
}
```

```
}
```

5.2.8 Drawing Bar Charts

A bar chart represents each data value by a rectangular bar whose **height is proportional to the value**. It is useful for comparing values of different categories, for example the marks obtained in different subjects. A loop draws one bar for every element of the data array.

Steps for drawing a bar chart

1. Store the values in an array and the category names in a String array.
2. Draw the x-axis and y-axis.
3. Decide the bar width, the gap between bars and the scale (pixels per unit).
4. Loop through the array. For each value compute the bar height and the position of the bar.
5. Draw the bar with `fillRect(x, y, barWidth, height)` and write the value and the category name with `drawString()`.

Formulas for the i-th bar

- $\text{height} = \text{value}[i] * \text{scale}$
- $x = \text{originX} + \text{gap} + i * (\text{barWidth} + \text{gap})$
- $y = \text{originY} - \text{height}$ (top edge of the bar, because y increases downwards)

Example 11 - Bar chart of marks

```
import java.applet.Applet;
import java.awt.*;

public class BarChartApplet extends Applet {
    int marks[] = {65, 80, 45, 90, 70};
    String subject[] = {"Maths", "Physics", "Chem", "Java", "English"};

    public void paint(Graphics g) {
        int originX = 40, originY = 230;
        int barWidth = 40, gap = 25;
        int scale = 2; // 1 mark = 2 pixels

        g.drawLine(originX, originY, originX, 20); // y-axis
        g.drawLine(originX, originY, originX + 340, originY); // x-axis

        for (int i = 0; i < marks.length; i++) {
            int height = marks[i] * scale;
            int x = originX + gap + i * (barWidth + gap);
            int y = originY - height;

            g.setColor(Color.blue);
            g.fillRect(x, y, barWidth, height); // the bar

            g.setColor(Color.black);
            g.drawString("" + marks[i], x + 10, y - 5); // value above the bar
            g.drawString(subject[i], x, originY + 15); // name below the axis
        }
    }
}
```

Subject	Marks	height = marks x 2	x	y = 230 - height
Maths	65	130	65	100

Subject	Marks	height = marks x 2	x	y = 230 - height
Physics	80	160	130	70
Chem	45	90	195	140
Java	90	180	260	50
English	70	140	325	90

Suitable APPLET tag: <applet code="BarChartApplet.class" width="400" height="270">.

Aspect	Line graph	Bar chart
Best for	Showing a trend or change over time	Comparing values of different categories
Main drawing methods	drawLine(), fillOval()	fillRect()
Role of the loop	Joins each point to the previous point	Draws one bar per data value
Conversion of values	y = originY - value x scale	height = value x scale; y = originY - height

Quick reference of drawing methods

Shape	Method	Parameters
Text	drawString	String, x, y
Line	drawLine	x1, y1, x2, y2
Rectangle	drawRect / fillRect	x, y, width, height
Rounded rectangle	drawRoundRect	x, y, width, height, arcWidth, arcHeight
Circle / ellipse	drawOval / fillOval	x, y, width, height
Arc / pie slice	drawArc / fillArc	x, y, width, height, startAngle, arcAngle
Polygon	drawPolygon / fillPolygon	int[] xPoints, int[] yPoints, nPoints (or a Polygon object)

Quick revision - 5.2 Graphics Programming

- Graphics is an abstract class in java.awt; the object is supplied to paint(Graphics g).
- Origin (0, 0) is at the top-left; x grows to the right and y grows downwards.
- drawOval with equal width and height gives a circle; the oval lies inside its bounding rectangle.
- In arcs, angle 0 is at 3 o'clock and positive angles go anticlockwise.
- drawPolygon closes the figure automatically; drawPolyline does not.
- Use loops to repeat shapes; for graphs and charts use y = originY - value x scale.

Chapter Practice

A. Multiple Choice Questions (15)

Choose the correct option for each question. The answer key is given at the end of this section.

Q1. Every applet must extend which class?

- (a) java.awt.Frame
- (b) java.applet.Applet
- (c) java.lang.Thread
- (d) java.io.Reader

Q2. Which method is called first when an applet is loaded?

- (a) start()
- (b) paint()
- (c) init()
- (d) destroy()

Q3. Which method is called every time the user returns to the page containing the applet?

- (a) start()
- (b) init()
- (c) destroy()
- (d) main()

Q4. Which method is called just before the applet is removed from memory?

- (a) stop()
- (b) init()
- (c) paint()
- (d) destroy()

Q5. Which attribute of the APPLET tag gives the name of the compiled applet class?

- (a) NAME
- (b) CODE
- (c) SRC
- (d) CLASS

Q6. Which method is used by an applet to read a parameter passed from the HTML page?

- (a) getValue()
- (b) getAttribute()
- (c) getParameter()
- (d) readParam()

Q7. What is the return type of the method getParameter()?

- (a) String
- (b) int
- (c) Object
- (d) void

- Q8.** Which JDK tool is used to test an applet without a web browser?
- (a) javac
 - (b) javap
 - (c) java
 - (d) appletviewer
- Q9.** Which of the following is NOT a restriction on an untrusted applet?
- (a) It cannot read or write files on the client
 - (b) It cannot display text in its window
 - (c) It cannot run programs on the client
 - (d) It can connect only to the host it came from
- Q10.** In the Graphics coordinate system, the origin (0, 0) is at the:
- (a) bottom-left corner
 - (b) centre of the window
 - (c) top-left corner
 - (d) top-right corner
- Q11.** Which method draws an unfilled (hollow) rectangle?
- (a) drawRect()
 - (b) fillRect()
 - (c) clearRect()
 - (d) paintRect()
- Q12.** To draw a circle of radius 40 pixels using drawOval(), the width and height should be:
- (a) 40 and 40
 - (b) 80 and 40
 - (c) 40 and 80
 - (d) 80 and 80
- Q13.** In drawArc(x, y, w, h, startAngle, arcAngle), the angle 0 degrees is at the:
- (a) 12 o'clock position
 - (b) 3 o'clock position
 - (c) 6 o'clock position
 - (d) 9 o'clock position
- Q14.** How many integer arrays are passed to drawPolygon(int[], int[], int)?
- (a) One array
 - (b) Three arrays
 - (c) Two arrays (x and y coordinates)
 - (d) None

Q15. Which method asks the system to redraw the applet by calling update() and then paint()?

- (a) repaint()
- (b) refresh()
- (c) redraw()
- (d) display()

Answer Key

Q. No.	Correct answer	Reason
1	(b) java.applet.Applet	An applet is a subclass of java.applet.Applet.
2	(c) init()	init() runs once, before any other life-cycle method.
3	(a) start()	start() is called after init() and again on each return to the page.
4	(d) destroy()	destroy() performs the final clean-up.
5	(b) CODE	CODE specifies the class file, e.g. CODE="HelloApplet.class".
6	(c) getParameter()	getParameter(String name) reads the value of a PARAM tag.
7	(a) String	It always returns a String (or null if the parameter is absent).
8	(d) appletviewer	appletviewer runs the applet in its own window.
9	(b) It cannot display text in its window	Displaying text and graphics in its own window is allowed.
10	(c) top-left corner	x increases to the right and y increases downward from the top-left corner.
11	(a) drawRect()	drawRect() draws the outline; fillRect() draws a solid rectangle.
12	(d) 80 and 80	Diameter = 2 x radius = 80, and width = height gives a circle.
13	(b) 3 o'clock position	Angles start at 3 o'clock and positive angles go anticlockwise.
14	(c) Two arrays (x and y coordinates)	One array holds the x-coordinates and another the y-coordinates, plus the point count.
15	(a) repaint()	repaint() schedules the redraw; paint() should not be called directly.

B. Broad Questions (10)

- Q1.** Explain the life cycle of an applet with a neat diagram. Describe the purpose of the methods `init()`, `start()`, `paint()`, `stop()` and `destroy()`, and write a program to demonstrate them.
- Q2.** What are local and remote applets? Differentiate between an applet and an application. State the security restrictions imposed on applets.
- Q3.** Explain the steps involved in preparing, building, compiling and running an applet. Write an applet that displays "Hello Java Applet" and the HTML file needed to run it.
- Q4.** Explain the `APPLET` tag with the syntax and the meaning of its attributes (`CODE`, `CODEBASE`, `WIDTH`, `HEIGHT`, `ALT`, `NAME`, `ALIGN`, `VSPACE`, `HSPACE`, `ARCHIVE`). How is an applet added to an HTML file?
- Q5.** How are parameters passed to an applet? Write an applet which accepts the name and the marks of a student through `PARAM` tags and displays them. Show the HTML file also.
- Q6.** What is the `Graphics` class? Explain the coordinate system used in Java graphics and describe any eight commonly used methods of the `Graphics` class with their syntax.
- Q7.** Describe the methods used for drawing lines and rectangles (`drawLine`, `drawRect`, `fillRect`, `clearRect`, `drawRoundRect`, `draw3DRect`). Write an applet that draws a line, a hollow rectangle, a filled rectangle and a rounded rectangle.
- Q8.** Explain how circles and ellipses are drawn in Java. Describe the `drawArc()` and `fillArc()` methods with the meaning of each argument, and write an applet that draws a pie chart with three slices.
- Q9.** How is a polygon drawn using the `Graphics` class? Explain `drawPolygon()`, `fillPolygon()` and the `Polygon` class with an applet that draws a triangle and a filled pentagon.
- Q10.** Explain how control loops are used in applets. Write an applet that uses a loop to draw a bar chart for five data values, with labels, and explain the formulas used to find the position and height of each bar. Also explain how a line graph is drawn.