Applets

- An Applet is a small application that are accessed through Internet
- It is developed and stored in a server machine.
- In response to the client request, Applet will send to the client machine.
- Java enabled browsers can display Applets
- Applet has limited acces to the resources in the client machine. Hence Applet is free from the risk of virus and guarantee data integrity

Applet Creation

- An Applet is a graphical user interface
- GUI is designed using Swing control
- Two versions of Windows controls
  - AWT control
  - SWING controls
- An Applet should be subclass of java.applet.Applet class and it should be **public**
- An Applet does not contains main() method

Applet Skeleton

- Applets are event driven
- An EVENT is a state of change of source
- Applets executes the statements based on events
- So programs are written within the methods equallent to particular event
- Event based methods in an Applet are
  - Public void init()
  - Public void start()
  - Public void paint()
  - Public void stop()
  - Public void destroy()

init( )

- The **init( )** method is the first method to be called. This is where you should initialize variables.
- This method is called only once during the run time of your applet.

start( )

- The **start( )** method is called after **init( )**. It is also called to restart an applet after it has been stopped.
- **start( )** is called each time an applet's HTML document is displayed onscreen. So, if a user leaves a web page and comes back, the applet resumes execution at **start( )**.

paint( )

- The **paint( )** method is called each time your applet's output must be redrawn.
- This situation can occur for several reasons. For example, the window in which the applet is running may be overwritten by another window and then uncovered.
- Or the applet window may be minimized and then restored. **paint( )** is also called when the applet begins execution.
- Whatever the cause, whenever the applet must redraw its output, **paint( )** is called.
- The **paint( )** method has one parameter of type **Graphics**. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running.

stop( )

- The **stop( )** method is called when a web browser leaves the HTML document containing the applet—when it goes to another page
- We can restart them when **start( )** is called if the user returns to the page.

destroy( )

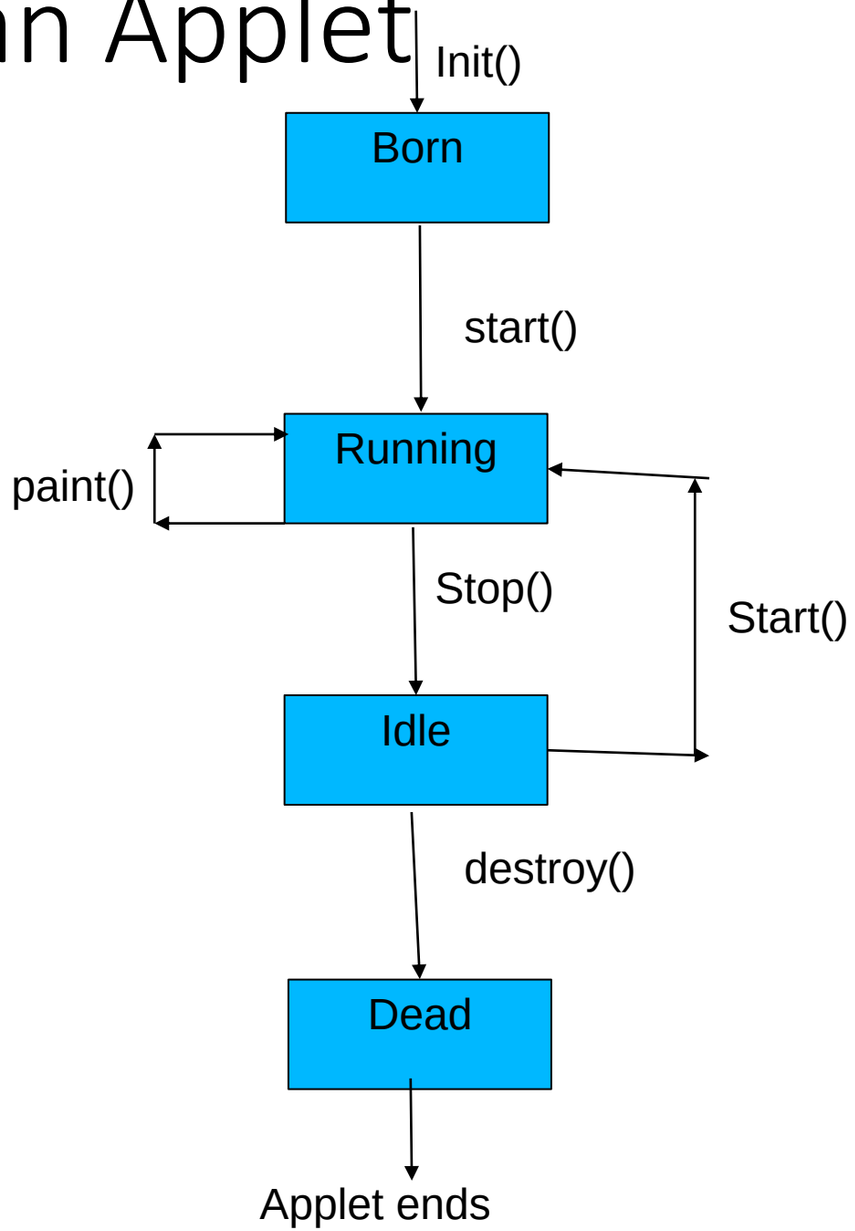
- The **destroy( )** method is called when the environment determines that your applet needs to be removed completely from memory.
- At this point, you should free up any resources the applet may be using.
- The **stop( )** method is always called before **destroy( )**.

Writing Applet Programs

- Applets are not standalone programs
- Applets are embedded within Web pages(HTML file)
- Sample Applet enabled HTML file

```
<html>
<body>
<applet code=”MyApplet” width=200 height=200></applet>
</body>
</html>
```

Life Cycle of an Applet



```
Import java.applet.*;
Import java.awt.*;
/*
<applet code=Applet1 width=200 height=150>
</applet> */
Public class Applet1 extends Applet
{
    public void paint(Graphics g)
    {
        g.drawstring(“Welcome”,20,40);
    }
}
```

- setBackground(Color.green) & getBackground()
- setForeground(Color.red) & getForeground()
- String getAppletInfo() – returns string that describes applet.
- URL getCodeBase() – return URL associated with the applet.
- getParameter(string pname) – returns the string associated with the parameter pname.
- Boolean isAlive() – returns true if the applet has been started.
- Void resize(int width, int height) – resizes the window according to the width and height.
- Void showStatus(String s) – displays the strings in the status window of the browser.

Passing Parameters to Applet

- The APPLET tag in HTML allows you to pass parameters to your applet.
- To retrieve a parameter, use the **getParameter() method. It returns the value of the specified** parameter in the form of a **String object.**

COLOR Class

- The Color class is a part of Java Abstract Window Toolkit(AWT) package. The Color class creates color by using the given RGBA values where **RGBA** stands for RED, GREEN, BLUE, ALPHA or using **HSB** value where HSB stands for HUE, SATURATION, BRIGHTNESS components.
- The value for individual components RGBA ranges from 0 to 255 or 0.0 to 0.1. The value of alpha determines the opacity of the color, where 0 or 0.0 stands fully transparent and 255 or 1.0 stands opaque.

- **Color(int r, int g, int b)** : Creates an opaque RGB color with the specified red, green, and blue values in the range (0 - 255).
- **Color(float r, float g, float b)** : creates a opaque color with specified RGB components(values are in range 0.0 – 0.1)
- **Color(int rgbValue)**: Creates an opaque RGB color with the specified combined RGB value consisting of the red component in bits 16-23, the green component in bits 8 – 15, and the blue component in bits 0-7.

- Hue, Saturation and Brightness(HSB) color model is an alternative to red-green-blue(RGB) for specifying particular colors.
- Hue is a wheel of color. The hue is specified with a number between 0.0 and 1.0 (the colors are approximately red, orange, yellow, green, blue, indigo, and violet).
- Saturation is another scale ranging from 0.0 to 1.0, representing light pastels to intense hues.
- Brightness values also range from 0.0 to 1.0, where 1 is bright white and 0 is black.
- Color supplies two methods that let you convert between RGB and HSB:

- static int HSBtoRGB(float hue, float saturation, float brightness)
- static float[ ] RGBtoHSB(int red, int green, int blue, float values[ ])
- HSBtoRGB( ) returns a packed RGB value compatible with the Color(int) constructor.
- RGBtoHSB( ) returns a float array of HSB values corresponding to RGB integers.
- If values is not null, then this array is given the HSB values and returned. Otherwise, a new array is created and the HSB values are returned in it.
- In either case, the array contains the hue at index 0, saturation at index 1, and brightness at index 2.

- To return red, green, and blue components of a color independently using getRed(), getGreen( ), and getBlue( ), shown here:
- int getRed( ) int getGreen( ) int getBlue( )
- Each of these methods returns the RGB color component found in the invoking Color object in the lower 8 bits of an integer.
- To return a packed, RGB representation of a color, use getRGB( ).
- Graphics objects are drawn in the current foreground color. You can change this color by calling the Graphics method setColor( ):
  - void setColor(Color newColor)
  - newColor specifies the new drawing color.
- To return current color by calling getColor( ), shown here:
  - Color getColor( )

Images in Applet

- Applet is mostly used in games and animation. For this purpose image is required to be displayed. The java.awt.Graphics class provide a method drawImage() to display the image.
- **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
- The java.applet.Applet class provides getImage() method that returns the object of Image.
- **public** Image getImage(URL u, String image)

Working with Graphics

- The Swing supports a rich assortment of graphics methods.
- All graphics are drawn relative to a window
- A *graphics context is encapsulated by the* ***Graphics class and is obtained in two ways:***
- It is passed to an applet when one of its various methods, such as **paint( )** is called.
- It is returned by the **getGraphics( ) method of Component**

Drawing Lines

- void drawLine(int *startX, int startY, int endX, int endY)*
- **drawLine( ) displays a line in the current drawing color that begins at *startX, startY and ends*** at *endX, endY.*

Drawing Rectangles

- void drawRect(int *top, int left, int width, int height)*
- void fillRect(int *top, int left, int width, int height)*
- void drawRoundRect(int *top, int left, int width, int height,* int *xDiam, int yDiam)*
- void fillRoundRect(int *top, int left, int width, int height,* int *xDiam, int yDiam) - The diameter of the rounding* arc along the X axis is specified by *xDiam. The diameter of the rounding arc along the Y axis* is specified by *yDiam.*
- Public void draw3DRect(int top, int left, int width, int height, boolean raised)
- Public void fill3DRect(int top, int left, int width, int height, boolean raised)
- Public void clearRect(int top, int left, int width, int height) – Clears a rectangular area with top left corner at(top,left) with the specified width height.

Drawing Ellipses and Circles

- void drawOval(int *top, int left, int width, int height)*
- void fillOval(int *top, int left, int width, int height)*
- The ellipse is drawn within a bounding rectangle

Drawing Arcs

- void drawArc(int *x, int y, int width, int height, int startAngle,* int *sweepAngle)*
- void fillArc(int *x, int y, int width, int height, int startAngle,* int *sweepAngle)*
- The arc is bounded by the rectangle whose upper-left corner is specified by *(x,y) and whose* width and height are specified by *width and height.*
- *The arc is drawn from startAngle through* the angular distance specified by *sweepAngle. Angles are specified in degrees.*

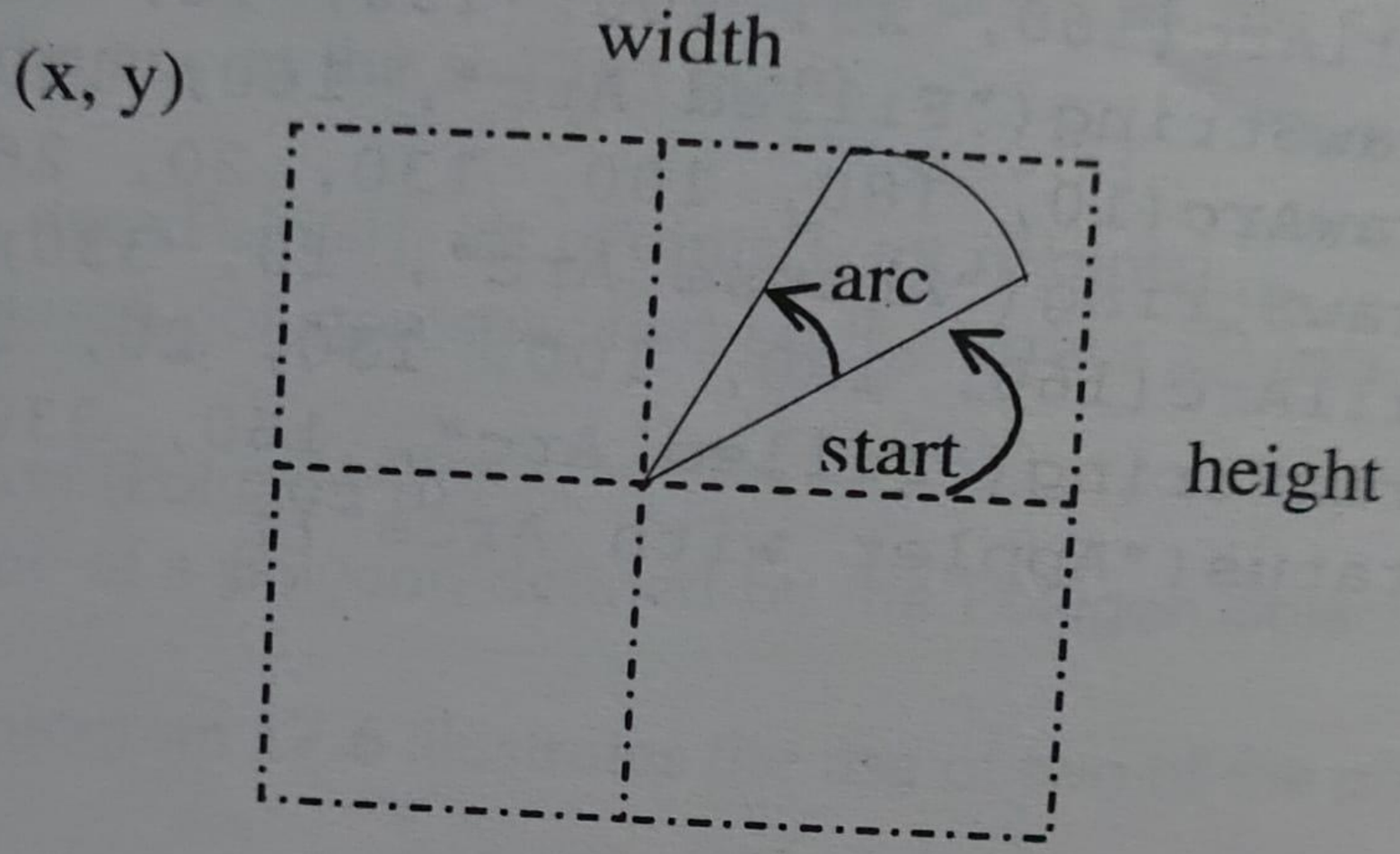


Fig. 17.6 Parameters Specifying an Arc

Drawing Polygons

- void drawPolygon(int *x[ ], int y[ ], int numPoints)*
- void fillPolygon(int *x[ ], int y[ ], int numPoints)*
- The polygon's endpoints are specified by the coordinate pairs contained within the *x and y* arrays.
- The number of points defined by *x and y is specified by numPoints.*

Fonts in Graphics

- Fonts loaded in the local computer system can be used for display.
- To set new fonts, methods in Graphics class are used.
- Fonts are available in Font class.
- The font family available in the local computer system can be obtained using getAvailableFontFamilyNames() method defined in GraphicsEnvironment class of java.awt package.
- Abstract String[] getAvailableFontFamilyNames() – Returns a String array containing the names of all font families available in the current Graphics Environment.
- To make use of this method, one need a GraphicsEnvironment object.
- To create GraphicsEnvironment object,
- Static GraphicsEnvironment getLocalGraphicsEnvironment() used, which returns local GraphicsEnvironment.

Setting Fonts

- Font(String name, int style, int size)
  - name – name of the font family
  - style – takes int values or the constants like Font.PLAIN, Font.BOLD, Font.ITALIC, or Font.ITALIC + Font.BOLD
  - size – is the font size
- Some of the methods defined Font class are on next slide.
- Public void setFont(Font bfont) – is defined in Graphics class to set font.

Table 17. 2 Some of the Methods Defined in Font Class

Method	Purpose of the Method
1. public String getFamily()	Returns the family name of this font
2. public String getName()	Returns the logical name of this font
3. public String getFontName()	Returns the font face name of this font Example : Serief. BOLD
4. public int getStyle()	Returns the style of the font PLAIN, BOLD, ITALIC, BOLD+ITALIC
5. public int getSize()	Returns an int representing the size of the font
6. public boolean isPlain()	Returns true, if this font has a PLAIN style, otherwise false
7. public boolean isBold()	Returns true if this font has a BOLD style, otherwise false
8. public boolean isItalic()	Returns true if this font has a ITALIC style, otherwise false

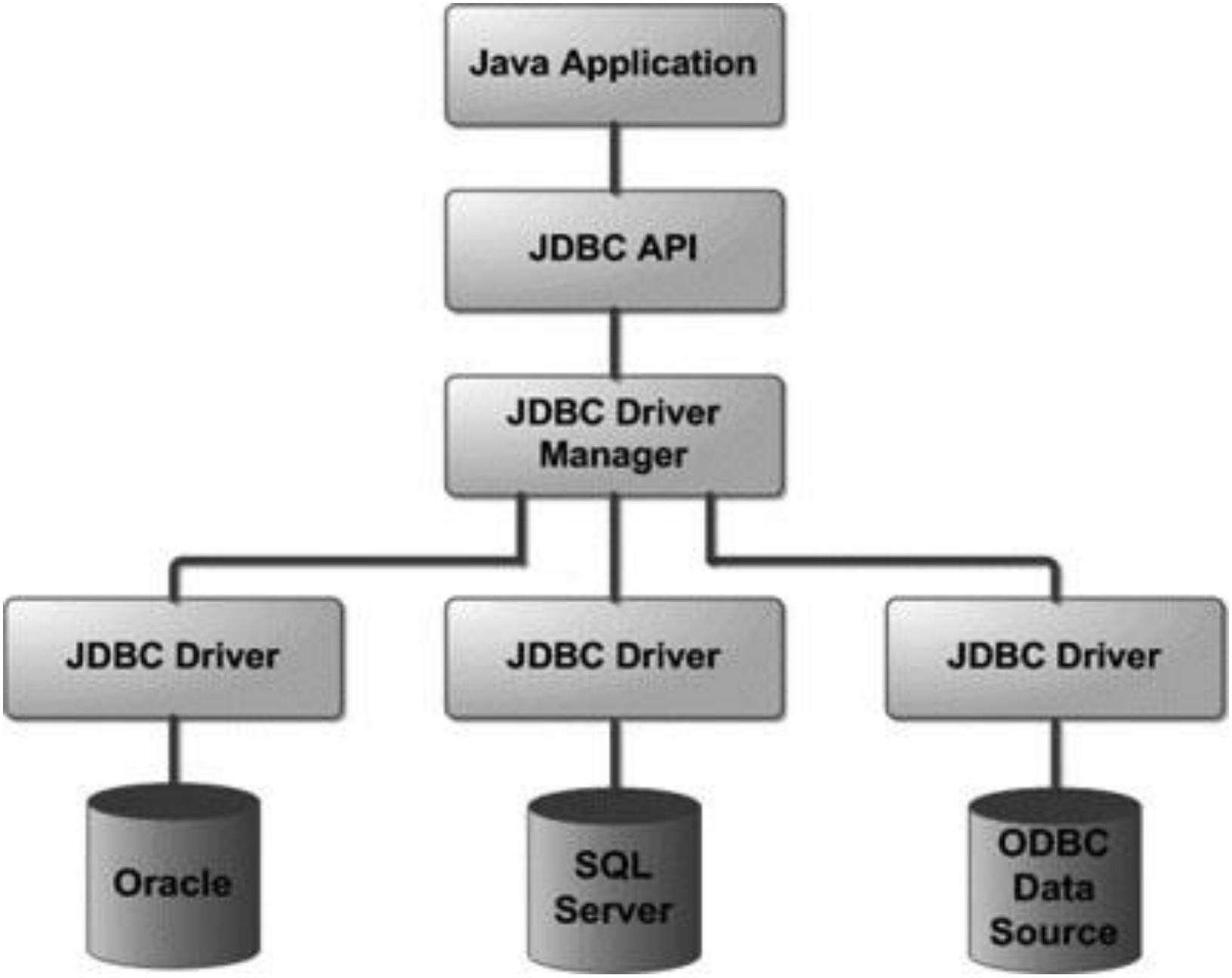
JDBC

- JDBC stands for **J**ava **D**ata**B**ase **C**onnectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.
- Java Application cannot directly communicate with a database to submit data & retrieve the results of queries. This is because a database can interpret only SQL statements & not java language statements.
- JDBC is a mechanism to translate java statements into SQL statements. JDBC is a Java API for executing SQL statements.

- The JDBC library includes APIs for each of the tasks commonly associated with database usage:
  - Making a connection to a database
  - Creating SQL or MySQL statements
  - Executing that SQL or MySQL queries in the database
  - Viewing & Modifying the resulting records

JDBC Architecture:

- The JDBC API supports both two-tier and three-tier processing models for database access but in general JDBC Architecture consists of two layers:
  - **JDBC API:** This provides the application-to-JDBC Manager connection.
  - **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.
- The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.
- The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.


Java Application
JDBC API
JDBC Driver Manager
JDBC Driver
JDBC Driver
JDBC Driver
Oracle
SQL Server
ODBC Data Source

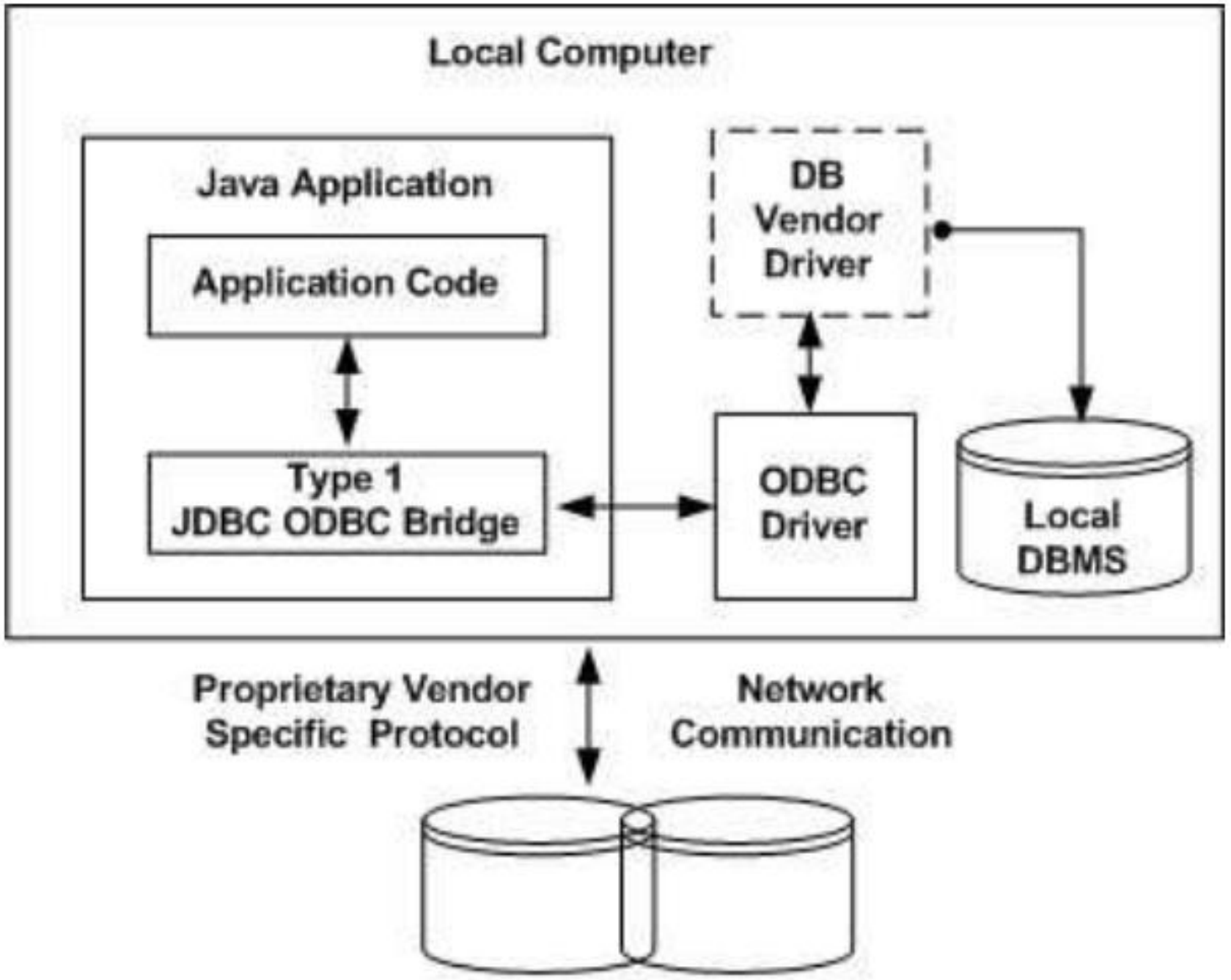
- **DriverManager:** This class is the traditional management layer of JDBC, working between the user and the drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver.
- **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects
- **Connection :** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
- **Statement :** You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures
- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- **SQLException:** This class handles any errors that occur in a database application.

JDBC Driver

- JDBC drivers implement the defined interfaces in the JDBC API for interacting with your database server.
- For example, using JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.
- **Type 1: JDBC-ODBC Bridge Driver**
- Type 2: Native-API-Partly-Java Driver
- Type 3: JDBC-NET-ALL-Java Driver
- Type 4: Native-Protocol-All-Java Driver

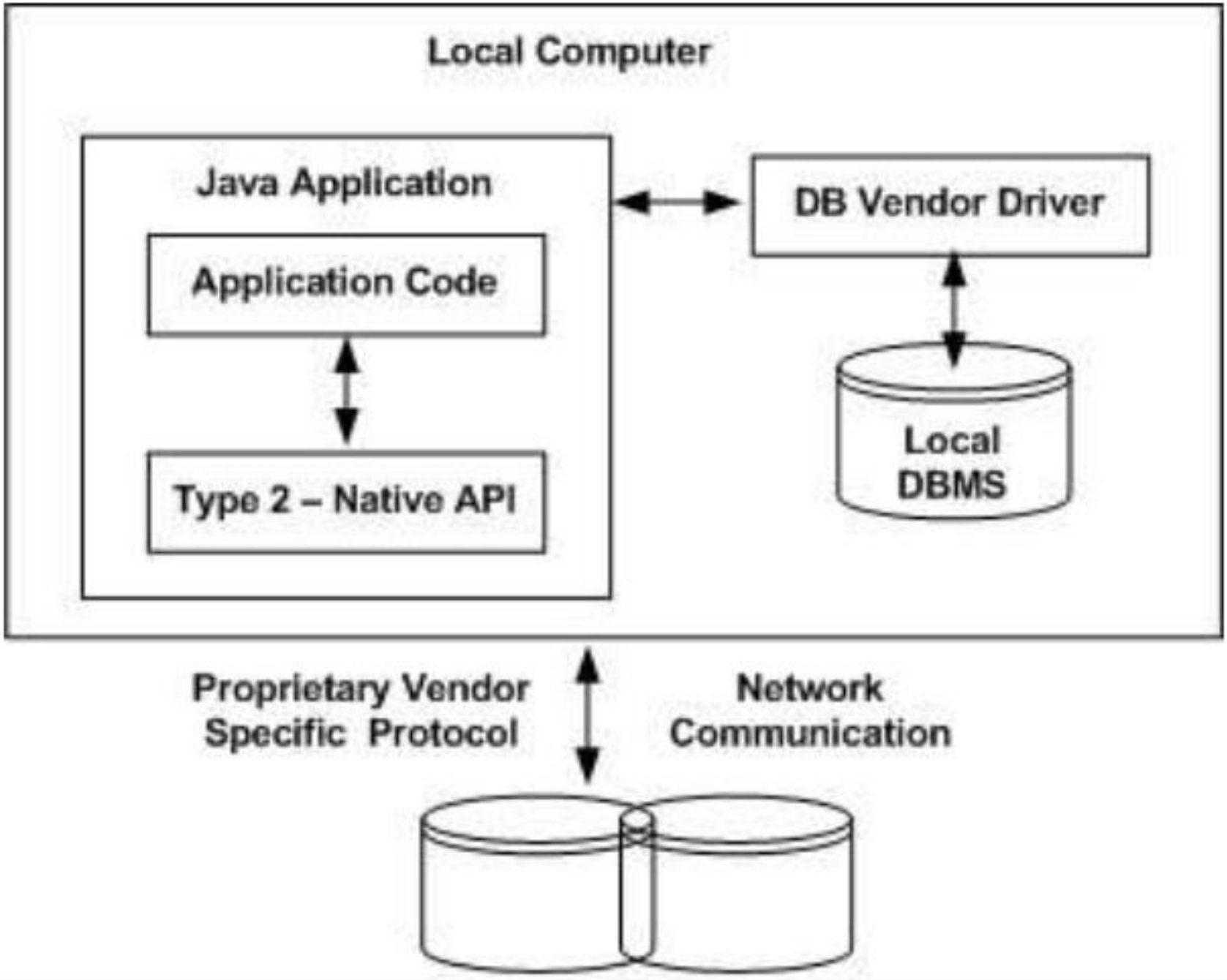
JDBC-ODBC Bridge Driver

- Type-1 driver or JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. Type-1 driver is also called Universal driver because it can be used to connect to any of the databases.
  - As a common driver is used in order to interact with different databases, the data transferred through this driver is not so secured.
  - The ODBC bridge driver is needed to be installed in individual client machines.
  - Type-1 driver isn’t written in java, that’s why it isn’t a portable driver.


Local Computer
Java Application
Application Code
Type 1
JDBC ODBC Bridge
DB
Vendor
Driver
ODBC
Driver
Local
DBMS
Proprietary Vendor
Specific Protocol
Network
Communication

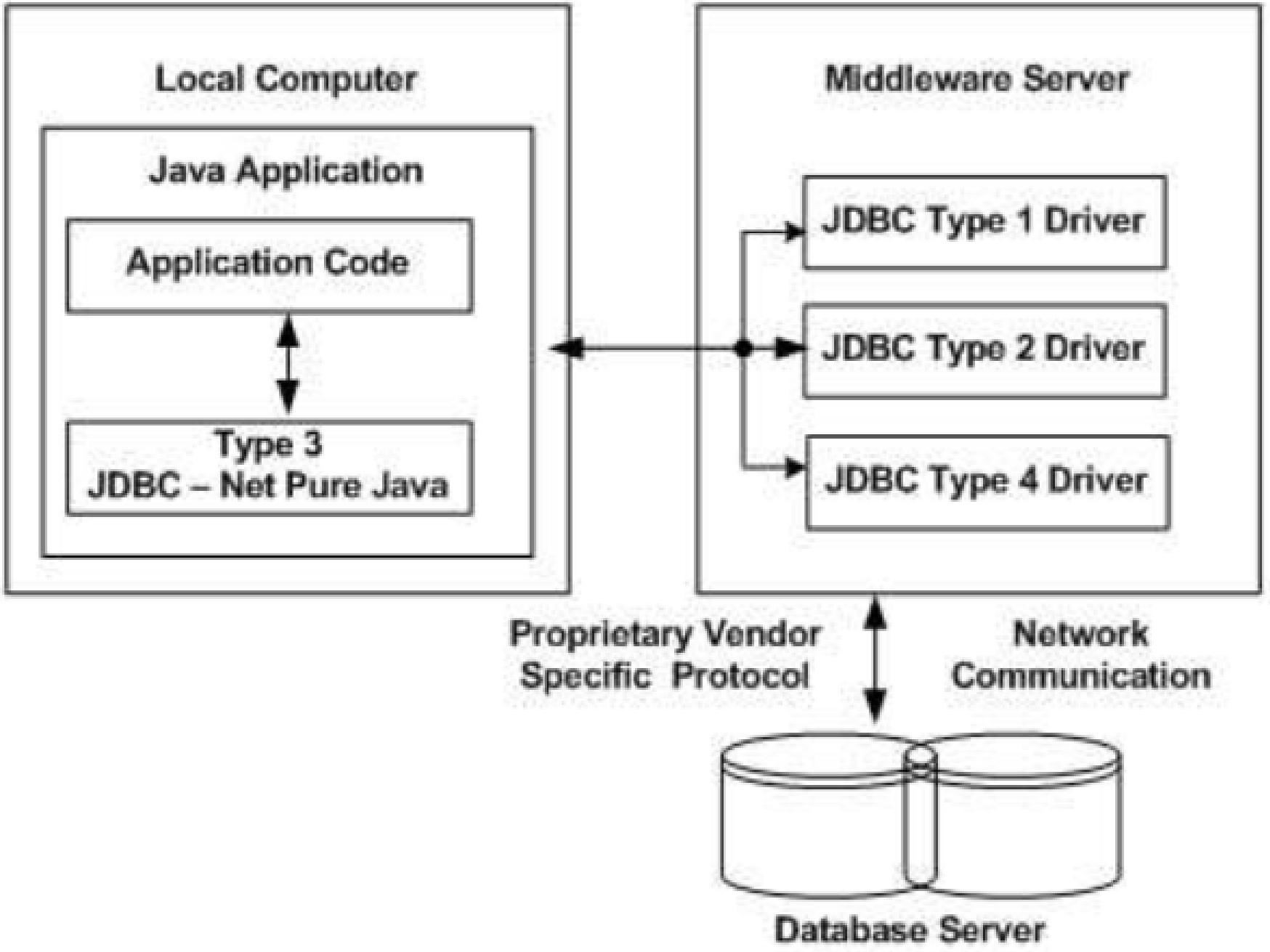
Native-API-Partly-Java Driver

- The Native API driver uses the client -side libraries of the database. This driver converts JDBC method calls into native calls of the database API. In order to interact with different database, this driver needs their local API, that's why data transfer is much more secure as compared to type-1 driver.
  - Driver needs to be installed separately in individual client machines
  - The Vendor client library needs to be installed on client machine.
  - Type-2 driver isn't written in java, that's why it isn't a portable driver


Local Computer
Java Application
Application Code
Type 2 – Native API
DB Vendor Driver
Local
DBMS
Proprietary Vendor
Specific Protocol
Network
Communication

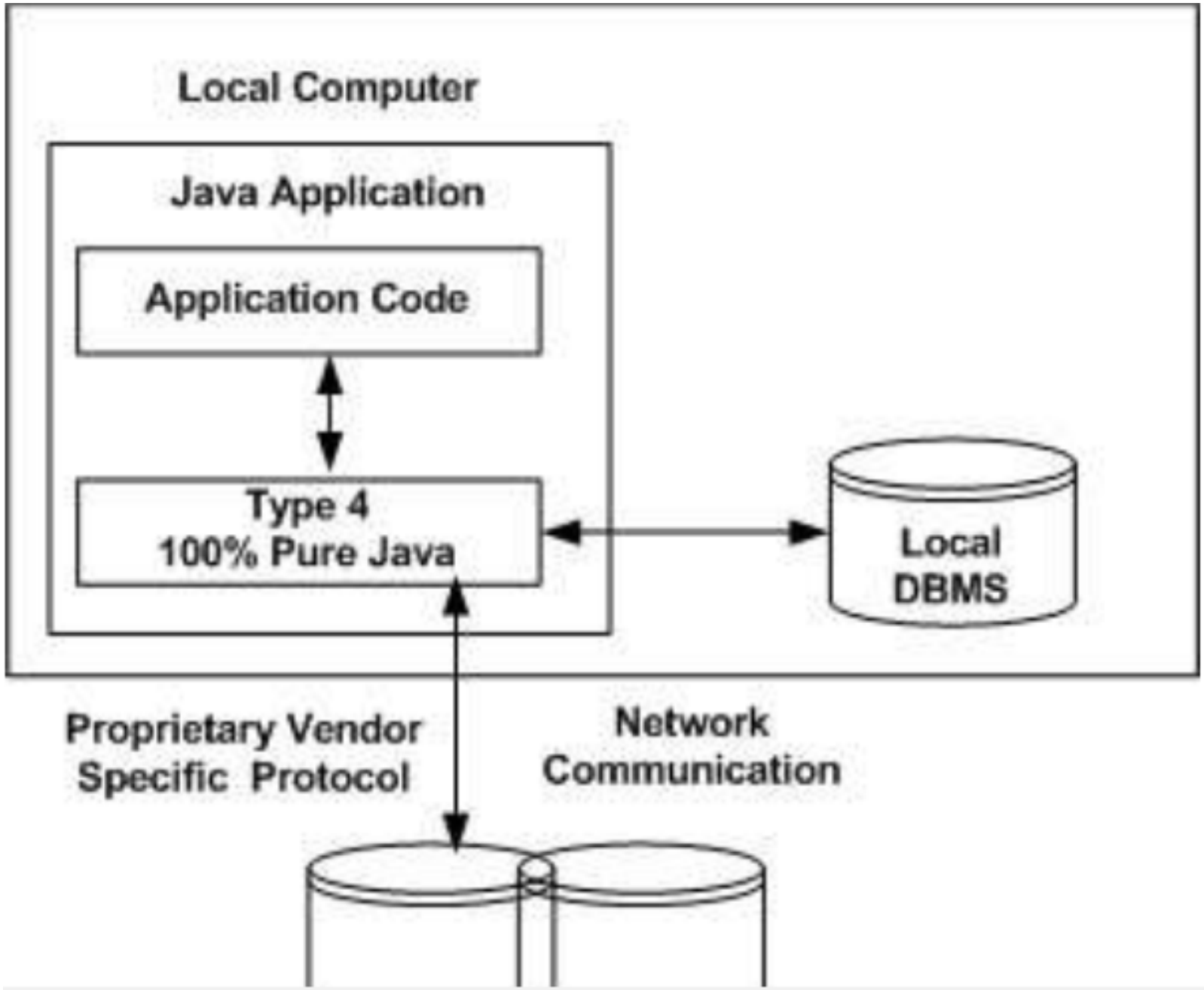
JDBC-NET-ALL-Java Driver

- In a Type 3 driver, a three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.
- This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.


Local Computer
Java Application
Application Code
Type 3
JDBC – Net Pure Java
Middleware Server
JDBC Type 1 Driver
JDBC Type 2 Driver
JDBC Type 4 Driver
Proprietary Vendor
Specific Protocol
Network
Communication
Database Server

Native-Protocol-All-Java Driver

- In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.
- This kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.


Local Computer
Java Application
Application Code
Type 4
100% Pure Java
Local
DBMS
Proprietary Vendor
Specific Protocol
Network
Communication

Using JDBC

- **Import the packages .** Requires that you include the packages containing the JDBC classes needed for database programming. Most often, using *import java.sql.** will suffice.
- **Register the JDBC driver .** Requires that you initialize a driver so you can open a communications channel with the database.
- **Open a connection .** Requires using the *DriverManager.getConnection()* method to create a Connection object, which represents a physical connection with the database.
- **Execute a query .** Requires using an object of type Statement for building and submitting an SQL statement to the database.
- **Extract data from result set .** Requires that you use the appropriate *ResultSet.getXXX()* method to retrieve the data from the result set.
- **Clean up the environment .** Requires explicitly closing all database resources versus relying on the JVM's garbage collection.

Import JDBC Packages:

- The **Import** statements tell the Java compiler where to find the classes you reference in your code and are placed at the very beginning of your source code.
- import java.sql.* ; // for standard JDBC programs

JDBC URL

- Database url. It have three components:
- <protocol> : <subprotocol> : subname
- **ProtocolName**:- protocol to access the database. It should be jdbc.
- **Sub-protocol**:- used to specify the type of database source, like oracle, Sybase etc.
- **Subname**:- specify the database server. Depends on subprotocol used.
- Subname may be network host name, database listener port number/name or database instance name.

Register JDBC Driver:

- The most common approach to register a driver is to use Java's Class.forName() method to dynamically load the driver's class file into memory, which automatically registers it. This method is preferable because it allows you to make the driver registration configurable and portable.

```
try {
   Class.forName("oracle.jdbc.driver.OracleDriver");
}
catch(ClassNotFoundException ex) {
   System.out.println("Error: unable to load driver class!");
   System.exit(1);
}
```

Database URL Formulation:

- The **getConnection(String url)** method of Java DriverManager class attempts to establish a connection to the database by using the given database URL. The appropriate driver from the set of registered JDBC drivers is selected.
- public static Connection getConnection(String url) throws SQLException
- getConnection(String url, Properties info)
- The **getConnection(String url, String user, String password)** method of Java DriverManager class attempts to establish a connection to the database by using the given database url. The appropriate driver from the set of registered JDBC drivers is selected.

RDBMS	JDBC driver name	URL format
MySQL	com.mysql.jdbc.Driver	jdbc:mysql://hostname/ databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@hostname:portNumber:databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	jdbc:db2:hostname:portNumber/databaseName
Sybase	com.sybase.jdbc.SybDriver	jdbc:sybase:Tds:hostname: portNumber/databaseName

Create Connection Object:

- **Using a database URL with a username and password:**
- String URL = "jdbc:oracle:thin:@amrood:1521:EMP";
- String USER = "username";
- String PASS = "password"
- Connection conn = DriverManager.getConnection(URL, USER, PASS);
- **Using only a database URL:**
- String URL = "jdbc:oracle:thin:username/password@amrood:1521:EMP";
- Connection conn = DriverManager.getConnection(URL);

- **Closing JDBC connections:**
- At the end of your JDBC program, it is required explicitly close all the connections to the database to end each database session. However, if you forget, Java's garbage collector will close the connection when it cleans up stale objects.
- conn.close();

Statements

- Once a connection is obtained we can interact with the database.
- Connection interface contains methods to create statements.
- The JDBC *Statement, CallableStatement, and PreparedStatement* interfaces define the methods and properties that enable you to send SQL or PL/SQL commands and receive data from your database.

Interfaces	Recommended Use
Statement	Use this for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.
PreparedStatement	Use this when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.
CallableStatement	Use this when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters.

- Connection interface has following methods:

1) public Statement createStatement(): creates a statement object that can be used to execute SQL queries.
2) CallableStatement prepareCall(String sql): creates a CallableStatement object for calling stored procedures.
3) public void setAutoCommit(boolean status): is used to set the commit status.By default it is true.
4) public void commit(): saves the changes made since the previous commit/rollback permanent.
5) public void rollback(): Drops all changes made since the previous commit/rollback.
6) public void close(): closes the connection and Releases a JDBC resources immediately.
7) PreparedStatement prepareStatement(String sql): creates a PreparedStatement object.

- Statement interface have several concrete methods to execute SQL statements.

1) public ResultSet executeQuery(String sql): is used to execute SELECT query. It returns the object of ResultSet.

2) public int executeUpdate(String sql): is used to execute specified query, it may be create, drop, insert, update, delete etc.

3) public boolean execute(String sql): is used to execute queries that may return multiple results. The result obtained is to be retreaved using getResultSet() method.

4) Void close(): Releases the Statement objects database and JDBC resources.

5) Int getMaxRows(): returns the maximum number of rows that the result set contains.

6) ResultSet getResultSet(): Retrieves the ResultSet generated by the execute() method.