JavaFX

JavaFX provides basic and advanced concepts of JavaFX.

JavaFX is a Java library that is used to develop **Desktop applications** as well as **Rich Internet Applications (RIA)**. The applications built in JavaFX, can run on multiple platforms including Web, Mobile and Desktops.

JavaFX includes all topics of JavaFX library such as Fundamentals, 2D Shapes, 3D Shapes, Effects, Animation, Text, Layouts, UI Controls, Transformations, Charts, JavaFX with CSS, JavaFX with Media etc.

What is JavaFX?

JavaFX is a Java library used to develop Desktop applications as well as Rich Internet Applications (RIA). The applications built in JavaFX, can run on multiple platforms including Web, Mobile and Desktops.

JavaFX is **intended to replace swing** in Java applications as a GUI framework. However, It provides more functionalities than swing. Like Swing, JavaFX also provides its own components and doesn't depend upon the operating system. It is lightweight and hardware accelerated. It supports various operating systems including Windows, Linux and Mac OS.

Feature	Description
Java Library	It is a Java library which consists of many classes and interfaces that are written in Java.
FXML	FXML is the XML based Declarative mark up language. The coding can be done in FXML to provide the more enhanced GUI to the user.
Scene Builder	Scene Builder generates FXML mark-up which can be ported to an IDE.
Web view	Web pages can be embedded with JavaFX applications. Web View uses WebKitHTML technology to embed web pages.
Built in UI controls	JavaFX contains Built-in components which are not dependent on operating system. The UI component are just enough to develop a full featured application.
CSS like styling	JavaFX code can be embedded with the CSS to improve the style of the application. We can enhance the view of our application with the simple knowledge of CSS.
Swing interoperability	The JavaFX applications can be embedded with swing code using the Swing Node class. We can update the existing swing application with the powerful features of JavaFX.
Canvas API	Canvas API provides the methods for drawing directly in an area of a JavaFX scene.
Rich Set of APIs	JavaFX provides a rich set of API's to develop GUI applications.
Integrated Graphics Library	An integrated set of classes are provided to deal with 2D and 3D graphics.
Graphics Pipeline	JavaFX graphics are based on Graphics rendered pipeline(prism). It offers smooth graphics which are hardware accelerated.
High Performance Media Engine	The media pipeline supports the playback of web multimedia on a low latency. It is based on a Gstreamer Multimedia framework.

JavaFX Line

In general, Line can be defined as the geometrical structure which joins two points (X1,Y1) and (X2,Y2) in a X-Y coordinate plane. JavaFX allows the developers to create the line on the GUI of a JavaFX application. JavaFX library provides the class Line which is the part of **javafx.scene.shape package**.

How to create a Line?

Follow the following instructions to create a Line.

- Instantiate the class javafx.scene.shape.Line.
- set the required properties of the class object.
- Add class object to the group

Properties

Line class contains various properties described below.

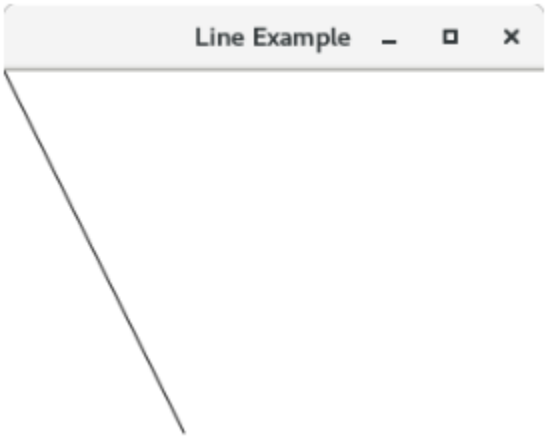
Property	Description	Setter Methods
endX	The X coordinate of the end point of the line	setEndX(Double)
endY	The y coordinate of the end point of the line	setEndY(Double)
startX	The x coordinate of the starting point of the line	setStartX(Double)
startY	The y coordinate of the starting point of the line	setStartY(Double)

Example 1

```
package application;

import javafx.application.Application;

import javafx.scene.Scene;
```

```
import javafx.scene.Group;
import javafx.scene.shape.Line;
import javafx.stage.Stage;
public class LineDrawingExamples extends Application{
    @Override
    public void start(Stage primaryStage) throws Exception {
        // TODO Auto-generated method stub
        Line line = new Line(); //instantiating Line class
        line.setStartX(0); //setting starting X point of Line
        line.setStartY(0); //setting starting Y point of Line
        line.setEndX(100); //setting ending X point of Line
        line.setEndY(200); //setting ending Y point of Line
        Group root = new Group(); //Creating a Group
        root.getChildren().add(line); //adding the class object //to the group
        Scene scene = new Scene(root,300,300);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Line Example");
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Output

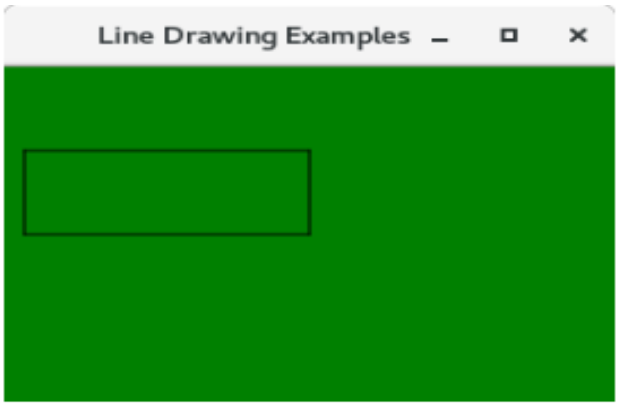


Example 2 : Creating Multiple Lines

```
package application;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.stage.Stage;
public class LineDrawingExamples extends Application{
    public static void main(String[] args) {
        launch(args);
    }
```

```
    @Override
    public void start(Stage primaryStage) throws Exception {
        // TODO Auto-generated method stub
        primaryStage.setTitle("Line Drawing Examples");
        Line line1 = new Line(10,50,150,50); //Line(startX,startY,endX,endY)
        Line line2 = new Line(10,100,150,100);
        Line line3 = new Line(10,50,10,100);
        Line line4 = new Line(150,50,150,100);
        Group root = new Group();
        root.getChildren().addAll(line1,line2,line3,line4);
        Scene scene = new Scene (root,300,200,Color.GREEN);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

}
```

Output:



JavaFX Rectangle

In general, Rectangles can be defined as the geometrical figure consists of four sides, out of which, the opposite sides are always equal and the angle between the two adjacent sides is 90 degree. A Rectangle with four equal sides is called square.

JavaFX library allows the developers to create a rectangle by instantiating **javafx.scene.shape.Rectangle** class.

Properties

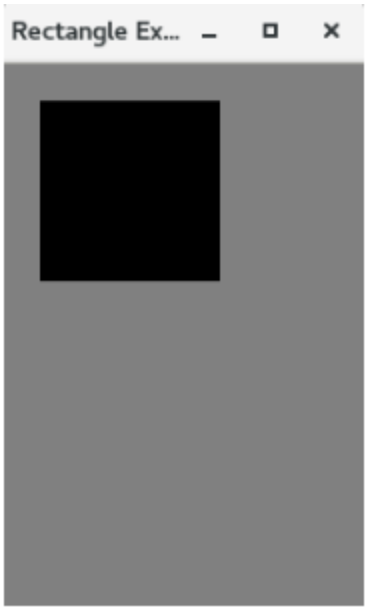
Property	Description	Setter Method
ArcHeight	Vertical diameter of the arc at the four corners of rectangle	setArcHeight(Double height)
ArcWidth	Horizontal diameter of the arc at the four corners of the rectangle	setArcWidth(Double Width)
Height	Defines the height of the rectangle	setHeight(Double height)
Width	Defines the width of the rectangle	setWidth(Double width)
X	X coordinate of the upper left corner	setX(Double X-value)
Y	Y coordinate of the upper left corner	setY(Double(Y-value)

Example 1:

```
package application;

import javafx.application.Application;

import javafx.scene.Group;

import javafx.scene.Scene;

import javafx.scene.paint.Color;

import javafx.scene.shape.Rectangle;

import javafx.stage.Stage;
```

```
public class Shape_Example extends Application{


    @Override
    public void start(Stage primaryStage) throws Exception {
        // TODO Auto-generated method stub
    primaryStage.setTitle("Rectangle Example");
    Group group = new Group(); //creating Group
    Rectangle rect=new Rectangle(); //instantiating Rectangle
    rect.setX(20); //setting the X coordinate of upper left //corner of rectangle
    rect.setY(20); //setting the Y coordinate of upper left //corner of rectangle
    rect.setWidth(100); //setting the width of rectangle
    rect.setHeight(100); // setting the height of rectangle
    group.getChildren().addAll(rect); //adding rectangle to the //group
    Scene scene = new Scene(group,200,300,Color.GRAY);
    primaryStage.setScene(scene);
    primaryStage.show();
}
public static void main(String[] args) {
    launch(args);
}


}
```

Output



Rounded Corner Rectangle

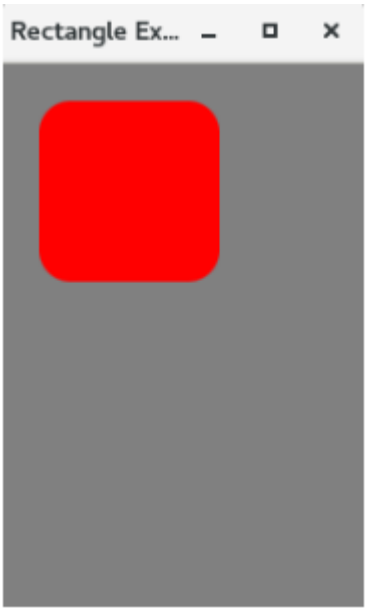
We can make the corners of the rectangle round by just calling the instance setter methods setArcHeight() and setArcWidth(). It sets the height and width of the arc at the corners of Rectangle. The following example implements Rounded corner rectangle.

Example:

```
package application;

import javafx.application.Application;

import javafx.scene.Group;

import javafx.scene.Scene;

import javafx.scene.paint.Color;

import javafx.scene.shape.Rectangle;
```

```
import javafx.stage.Stage;
public class Shape_Example extends Application{
    @Override
    public void start(Stage primaryStage) throws Exception {
        // TODO Auto-generated method stub
    primaryStage.setTitle("Rectangle Example");
    Group group = new Group();
    Rectangle rect=new Rectangle();
    rect.setX(20);
    rect.setY(20);
    rect.setWidth(100);
    rect.setHeight(100);
    rect.setArcHeight(35);
    rect.setArcWidth(35);
    rect.setFill(Color.RED);
    group.getChildren().addAll(rect);
    Scene scene = new Scene(group,200,300,Color.GRAY);
    primaryStage.setScene(scene);
    primaryStage.show();
}
public static void main(String[] args) {
    launch(args);
}  }
```

Output



JavaFX Cirlce

A circle is a special type of ellipse with both of the focal points at the same position. Its horizontal radius is equal to its vertical radius. JavaFX allows us to create Circle on the GUI of any application by just instantiating **javafx.scene.shape.Circle** class. Just set the class properties by using the instance setter methods and add the class object to the Group.

Properties

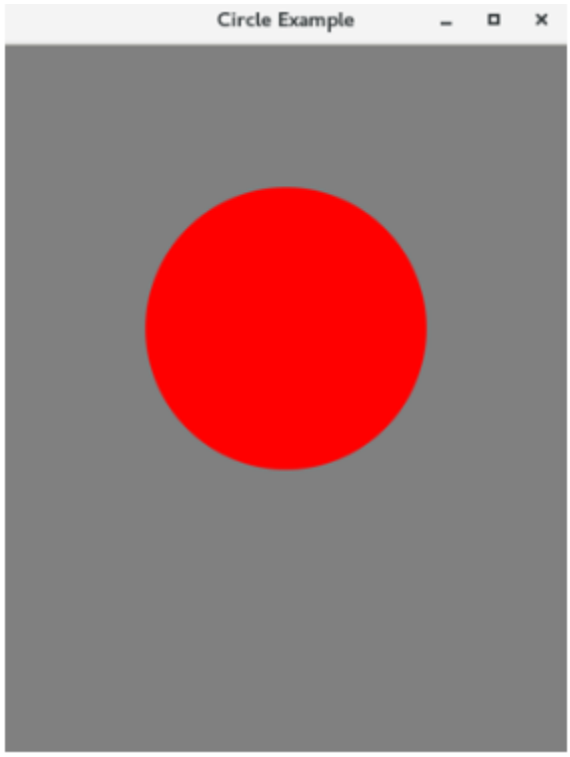
The class properties along with the setter methods and their description are given below in the table.

Property	Description	Setter Methods
centerX	X coordinate of the centre of circle	setCenterX(Double value)
centerY	Y coordinate of the centre of circle	setCenterY(Double value)
radious	Radius of the circle	setRadius(Double value)

Example

```
package application;

import javafx.application.Application;

import javafx.scene.Group;

import javafx.scene.Scene;

import javafx.scene.paint.Color;

import javafx.scene.shape.Circle;

import javafx.stage.Stage;
```

```
public class Shape_Example extends Application{


    @Override
    public void start(Stage primaryStage) throws Exception {
        // TODO Auto-generated method stub
    primaryStage.setTitle("Circle Example");
    Group group = new Group();
    Circle circle = new Circle();
    circle.setCenterX(200);
    circle.setCenterY(200);
    circle.setRadius(100);
    circle.setFill(Color.RED);
    group.getChildren().addAll(circle);
    Scene scene = new Scene(group,400,500,Color.GRAY);
    primaryStage.setScene(scene);
    primaryStage.show();
}
public static void main(String[] args) {
    launch(args);
}
}
```


Circle Example

JavaFX Text

In some of the cases, we need to provide the text based information on the interface of our application. JavaFX library provides a class named **javafx.scene.text.Text** for this purpose. This class provides various methods to alter various properties of the text. We just need to instantiate this class to implement text in our application.

Properties

The properties of JavaFX Text are described in the table below.

Property	Description	Setter Methods
boundstype	This property is of object type. It determines the way in which the bounds of the text is being calculated.	setBoundsType(TextBoundsType value)
font	Font of the text.	setFont(Font value)
fontsmoothingType	Defines the requested smoothing type for the font.	setFontSmoothingType(FontSmoothingType value)
linespacing	Vertical space in pixels between the lines. It is double type property.	setLineSpacing(double spacing)
strikethrough	This is a boolean type property. We can put a line through the text by setting this property to true.	setStrikeThrough(boolean value)
textalignment	Horizontal Text alignment	setTextAlignment(TextAlignment value)
textorigin	Origin of text coordinate system in local coordinate system.	setTextOrigin(VPos value)
text	It is a string type property. It defines the text string which is to be displayed.	setText(String value)
underline	It is a boolean type property. We can underline the text by setting this property to true.	setUnderLine(boolean value)
wrappingwidth	Width limit for the text from where the text is to be wrapped. It is a double type property.	setWrappingWidth(double value)
x	X coordinate of the text	setX(double value)
y	Y coordinate of the text	setY(double value)

Creating a text node

The class **javafx.scene.text**.Text needs to be instantiated in order to create the text node. Use the setter method setText(string) to set the string as a text for the text class object. Follow the syntax given below to instantiate the Text class.

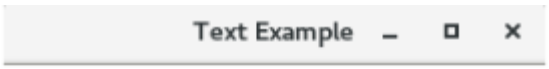
```
Text <text_Object> = new Text();
text.setText(<string-text>);
```

Example

The following example illustrates the Text class. Here, we are not setting the positions for the text therefore the text will be displayed to the centre of the screen.

```
package application;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.text.Text;
import javafx.stage.Stage;
public class TextExample extends Application{
@Override
public void start(Stage primaryStage) throws Exception {
   // TODO Auto-generated method stub
   Text text = new Text();
```

```
    text.setText("Hello !! Welcome to JavaTPoint");
    StackPane root = new StackPane();
    Scene scene = new Scene(root,300,400);
    root.getChildren().add(text);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Text Example");
    primaryStage.show();
}
public static void main(String[] args) {
    launch(args);

}
}
```



Hello !! Welcome to JavaTPoint

Font and position of the Text

JavaFX enables us to apply various fonts to the text nodes. We just need to set the property font of the Text class by using the setter method setFont(). This method accepts the object of Font class. The class Font belongs the package **javafx.scene.text**. It contains a static method named font(). This returns an object of Font type which will be passed as an argument into the setFont() method of Text class. The method Font.font() accepts the following parameters.

Family: it represents the family of the font. It is of string type and should be an appropriate font family present in the system.

Weight: this Font class property is for the weight of the font. There are 9 values which can be used as the font weight. The values are FontWeight.BLACK, BOLD, EXTRA_BOLD, EXTRA_LIGHT, LIGHT, MEDIUM, NORMAL, SEMI_BOLD, THIN.

Posture: this Font class property represents the posture of the font. It can be either FontPosture.ITALIC or FontPosture.REGULAR.

Size: this is a double type property. It is used to set the size of the font.

The Syntax of the method setFont() is given below.

<text_object>.setFont(Font.font(<String font_family>, <FontWeight>, <FontPosture>, <FontSize>)

Example

```
package application;

import javafx.application.Application;

import javafx.scene.Group;
```

```
import javafx.scene.Scene;
import javafx.scene.text.Font;
import javafx.scene.text.FontPosture;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;
public class TextExample extends Application{
@Override
public void start(Stage primaryStage) throws Exception {
    // TODO Auto-generated method stub
    Text text = new Text();

    text.setX(100);
    text.setY(20);
    text.setFont(Font.font("Abyssinica SIL",FontWeight.BOLD,FontPosture.REGULAR,20));
    text.setText("Welcome to JavaTPoint");
    Group root = new Group();
    Scene scene = new Scene(root,500,200);
    root.getChildren().add(text);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Text Example");
    primaryStage.show();
}
```

```
publicstaticvoid main(String[] args) {
   launch(args);

}
}
```

Applying Stroke and Color to Text

Stroke means the padding at the boundary of the text. JavaFX allows us to apply stroke and colors to the text. javafx.scene.text.Text class provides a method named setStroke() which accepts the Paint class object as an argument. Just pass the color which will be painted on the stroke. We can also set the width of the stroke by passing a width value of double type into setStrokeWidth() method. To set the color of the Text, javafx.scene.text.Text class provides another method named setFill(). We just need to pass the color which is to be filled in the text.

Example

The following example illustrates the functions of above mentioned methods.

```
package application;

import javafx.application.Application;

import javafx.scene.Group;

import javafx.scene.Scene;

import javafx.scene.paint.Color;

import javafx.scene.text.Font;

import javafx.scene.text.FontPosture;

import javafx.scene.text.FontWeight;

import javafx.scene.text.Text;

import javafx.stage.Stage;

public class TextExample extends Application{

@Override

publicvoid start(Stage primaryStage) throws Exception {

   // TODO Auto-generated method stub

   Text text = new Text();


   text.setX(100);

   text.setY(20);

   text.setFont(Font.font("Abyssinica SIL",FontWeight.BOLD,FontPosture.REGULAR,25));

   text.setFill(Color.BLUE);// setting colour of the text to blue
```

```
    text.setStroke(Color.BLACK); // setting the stroke for the text
    text.setStrokeWidth(1); // setting stroke width to 2
    text.setText("Welcome to JavaTPoint");
    Group root = new Group();
    Scene scene = new Scene(root,500,200);
    root.getChildren().add(text);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Text Example");
    primaryStage.show();
}
public static void main(String[] args) {
    launch(args);

}
}
```

Applying Decorations to the text

We can apply the decorations to the text by setting the properties strikethrough and underline of javafx.scene.text.Text class. The syntax of both the methods is given below.

```
<TextObject>.setStrikeThrough(Boolean value) //pass true to put a line across the text

<TextObject>.setUnderLine(Boolean value) //pass true to underline the text
```

We can also apply JavaFX Effects to the Text class objects. We will discuss JavaFX effects in the upcoming chapters.

Example

```
package application;

import javafx.application.Application;

import javafx.scene.Group;

import javafx.scene.Scene;

import javafx.scene.text.Font;

import javafx.scene.text.Text;

import javafx.stage.Stage;

public class TextExample extends Application{

@Override

public void start(Stage primaryStage) throws Exception {

    // TODO Auto-generated method stub
```

```
    Text text = new Text();


    text.setX(100);
    text.setY(40);
    text.setFont(Font.font("Liberation Serif",25));
    text.setText("Hello !!");
    text.setStrikethrough(true);
    Text text1=new Text();
    text1.setX(100);
    text1.setY(140);
    text1.setText("Welcome to JavaTPoint!");
    text1.setFont(Font.font("Liberation Serif",25));
    text1.setUnderline(true);
    Group root = new Group();
    Scene scene = new Scene(root,500,200);
    root.getChildren().addAll(text,text1);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Text Example");
    primaryStage.show();
}
public static void main(String[] args) {
    launch(args);
```

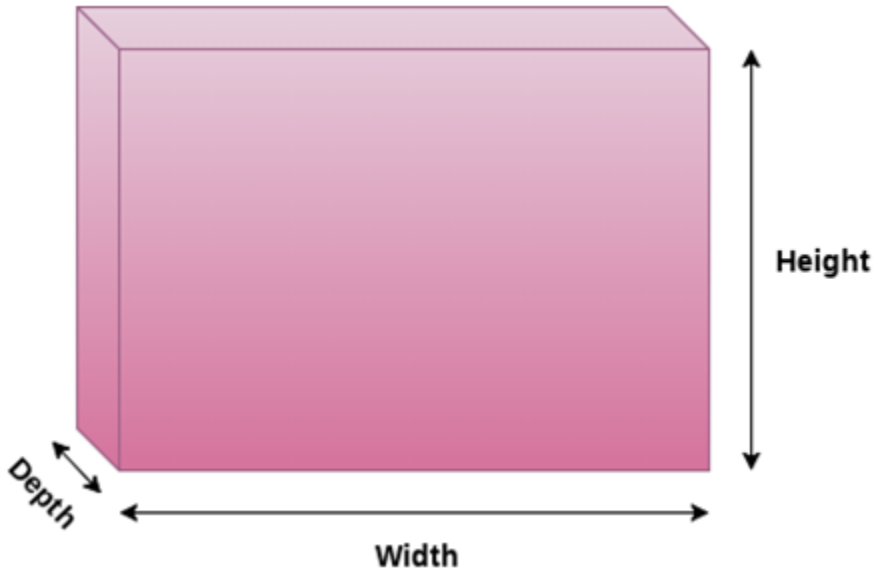
```
}

}
```


Text Example
Hello !!
Welcome to JavaTPoint!

JavaFX Box

In general, a box can be defined as the three dimensional shape having all the faces in the rectangular shape. Boxes are generally the cuboid having three dimensions height, depth and width. In JavaFX, box is represented by the class javafx.scene.shape.Box. We just need to instantiate this class in order to create the box.

The height, width and depth of a cube (box) is shown in the following image.



Properties

The class contains various properties that are described in the following table.

Property	Description	Setter Methods
depth	It is a double type property. It represents the z-dimension of the Box.	setDepth(double value)
height	It is a double type property. It represents the Y-dimension of the Box.	setHeight(double value)
width	It is a double type property. It represents the X-dimension of the Box.	setWidth(double value)

Constructors

The class contains two constructors described below.

public Box(): creates the instance of Box class with the default parameters.

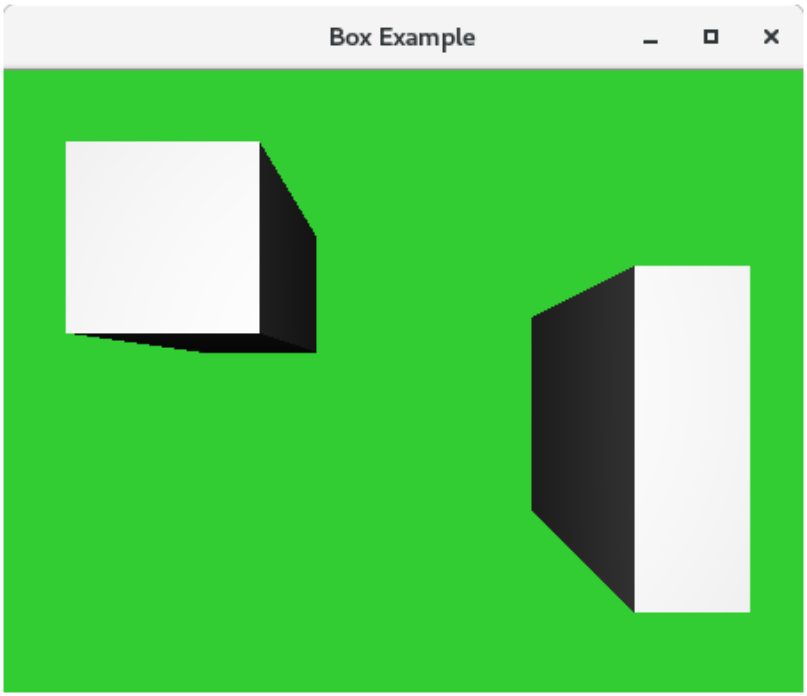
public Box(double width, double height, double depth) : creates the instance of Box class with the specified dimensions

Example

```
package application;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.PerspectiveCamera;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Box;
import javafx.stage.Stage;
public class BoxExample extends Application{
@Override
public void start(Stage primaryStage) throws Exception {
    // TODO Auto-generated method stub
    //Creating Boxes
    Box box1 = new Box();
```

```
Box box2 = new Box();
//Setting properties for second box
box2.setTranslateX(450);
box2.setTranslateY(300);
box2.setTranslateZ(100);
box2.setHeight(150);
box2.setWidth(50);
box2.setDepth(400);
//Setting properties for first box
box1.setHeight(100);
box1.setWidth(100);
box1.setDepth(400);
box1.setTranslateX(200);
box1.setTranslateY(200);
box1.setTranslateZ(200);
//Setting the perspective camera
PerspectiveCamera camera = new PerspectiveCamera();
camera.setTranslateX(100);
camera.setTranslateY(100);
camera.setTranslateZ(50);
//Configuring Group, Scene and Stage
Group root = new Group();
root.getChildren().addAll(box1,box2);
```

```
    Scene scene = new Scene(root,450,350,Color.LIMEGREEN);
    scene.setCamera(camera);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Box Example");
    primaryStage.show();
}
public static void main(String[] args) {
    launch(args);
}
}
```



Java Networking

Java Networking is a concept of connecting two or more computing devices together so that we can share resources.

Java socket programming provides facility to share data between different computing devices.

Java Networking Terminology

The widely used java networking terminologies are given below:

- IP Address
- Protocol
- Port Number
- MAC Address
- Connection-oriented and connection-less protocol
- Socket

1) IP Address

IP address is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of octets that range from 0 to 255.

It is a logical address that can be changed.

2) Protocol

A protocol is a set of rules basically that is followed for communication. For example:

TCP

FTP

Telnet

SMTP

POP etc.

3) Port Number

The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications.

The port number is associated with the IP address for communication between two applications.

4) MAC Address

MAC (Media Access Control) Address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC.

5) Connection-oriented and connection-less protocol

In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP.

But, in connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.

6) Socket

A socket is an endpoint between two way communication.

Java Socket Programming

Java Socket programming is used for communication between the applications running on different JRE.

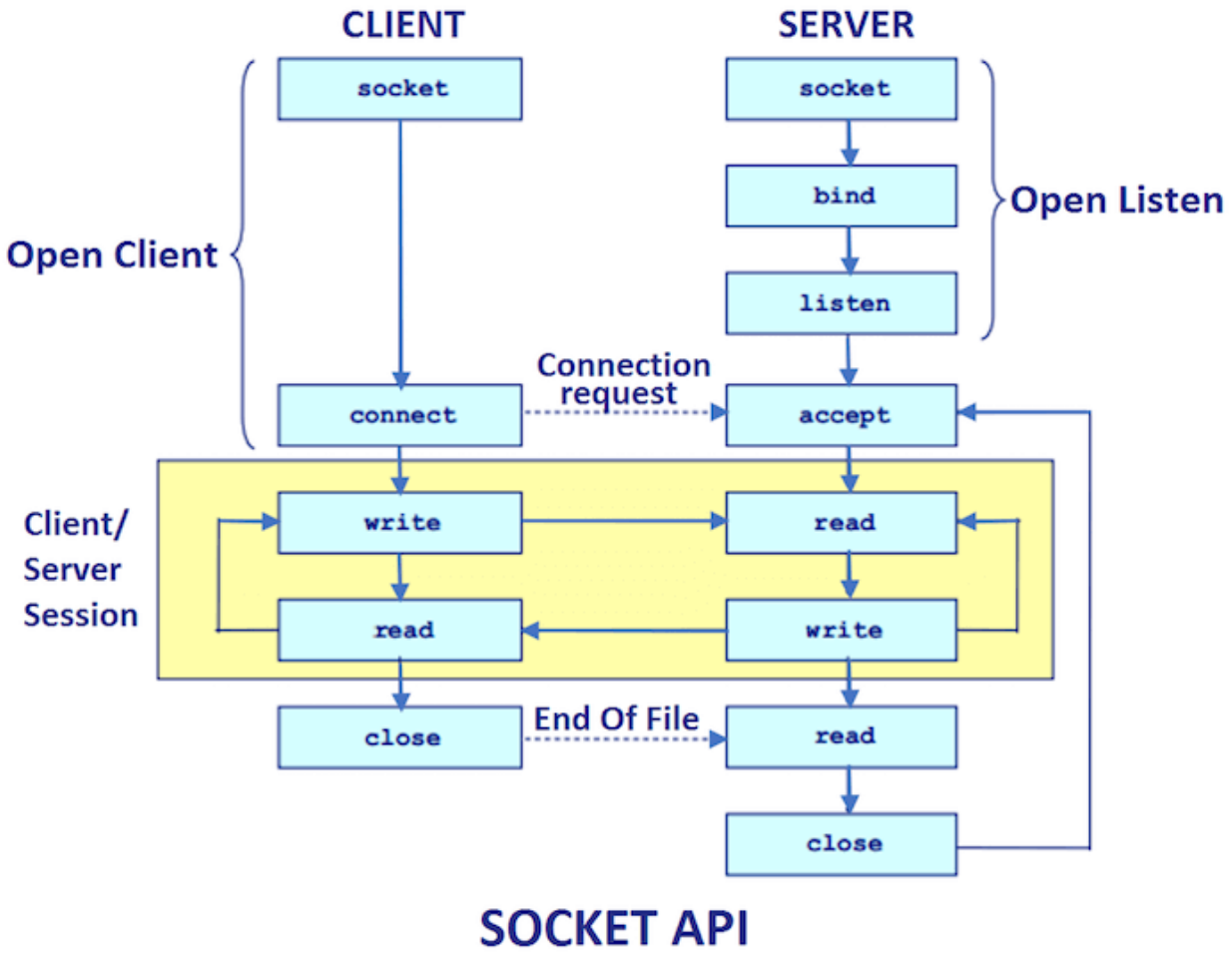
Java Socket programming can be connection-oriented or connection-less.

Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

The client in socket programming must know two information:

- IP Address of Server, and
- Port number.

Here, we are going to make one-way client and server communication. In this application, client sends a message to the server, server reads the message and prints it. Here, two classes are being used: Socket and ServerSocket. The Socket class is used to communicate client and server. Through this class, we can read and write message. The ServerSocket class is used at server-side. The accept() method of ServerSocket class blocks the console until the client is connected. After the successful connection of client, it returns the instance of Socket at server-side.



Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

Important methods

Method	Description
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

ServerSocket class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

Important methods

Method	Description
1) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket.

Java Socket Programming

Creating Server:

To create the server application, we need to create the instance of ServerSocket class. Here, we are using 6666 port number for the communication between the client and server. You may also choose any other port number. The accept() method waits for the client. If clients connects with the given port number, it returns an instance of Socket.

```
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection and waits for the client
```

Creating Client:

To create the client application, we need to create the instance of Socket class. Here, we need to pass the IP address or hostname of the Server and a port number. Here, we are using "localhost" because our server is running on same system.

```
Socket s=new Socket("localhost",6666);
```

Let's see a simple of Java socket programming where client sends a text and server receives and prints it.

File: MyServer.java

```
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection
DataInputStream dis=new DataInputStream(s.getInputStream());
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
ss.close();
}catch(Exception e){System.out.println(e);}
}
}
```

File: MyClient.java

```
import java.io.*;
import java.net.*;
```

```
public class MyClient {
public static void main(String[] args) {
try{
Socket s=new Socket("localhost",6666);
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
dout.flush();
dout.close();
s.close();
}catch(Exception e){System.out.println(e);}
}
}
```

Example of Java Socket Programming (Read-Write both side)

In this example, client will write first to the server then server will receive and print the text. Then server will write to the client and client will receive and print the text. The step goes on.

File: MyServer.java

```
import java.net.*;
import java.io.*;
class MyServer{
public static void main(String args[])throws Exception{
```

```
ServerSocket ss=new ServerSocket(3333);
Socket s=ss.accept();
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

String str="",str2="";
while(!str.equals("stop")){
str=din.readUTF();
System.out.println("client says: "+str);
str2=br.readLine();
dout.writeUTF(str2);
dout.flush();
}
din.close();
s.close();
ss.close();
}}
```

File: MyClient.java

```
import java.net.*;
import java.io.*;
class MyClient{
```

```
public static void main(String args[])throws Exception{
Socket s=new Socket("localhost",3333);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

String str="",str2="";
while(!str.equals("stop")){
str=br.readLine();
dout.writeUTF(str);
dout.flush();
str2=din.readUTF();
System.out.println("Server says: "+str2);
}

dout.close();
s.close();
}}
```

Java URL

The Java URL class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web. For example:

A URL contains many information:

Protocol: In this case, http is the protocol.

Server name or IP Address: In this case, www.javatpoint.com is the server name.

Port Number: It is an optional attribute. If we write http//ww.javatpoint.com:80/sonoojaiswal/ , 80 is the port number. If port number is not mentioned in the URL, it returns -1.

File Name or directory name: In this case, index.jsp is the file name.

Constructors of Java URL class

URL(String spec)

Creates an instance of a URL from the String representation.

URL(String protocol, String host, int port, String file)

Creates an instance of a URL from the given protocol, host, port number, and file.

URL(String protocol, String host, int port, String file, URLStreamHandler handler)

Creates an instance of a URL from the given protocol, host, port number, file, and handler.

URL(String protocol, String host, String file)

Creates an instance of a URL from the given protocol name, host name, and file name.

URL(URL context, String spec)

Creates an instance of a URL by parsing the given spec within a specified context.

URL(URL context, String spec, URLStreamHandler handler)

Creates an instance of a URL by parsing the given spec with the specified handler within a given context.

Example of Java URL class

```
//URLDemo.java
import java.net.*;
public class URLDemo{
public static void main(String[] args){
try{
URL url=new URL("http://www.javatpoint.com/java-tutorial");

System.out.println("Protocol: "+url.getProtocol());
System.out.println("Host Name: "+url.getHost());
System.out.println("Port Number: "+url.getPort());
System.out.println("File Name: "+url.getFile());
```

```
}catch(Exception e){System.out.println(e);}
}
}
```

Output:

Protocol: http

Host Name: www.javatpoint.com

Port Number: -1

File Name: /java-tutorial

Let us see another example URL class in Java.

```
//URLDemo.java
import java.net.*;
public class URLDemo{
public static void main(String[] args){
try{
URL url=new URL("https://www.google.com/search?q=javatpoint&oq=javatpoint&sourceid=chrome&ie=UTF-8");
```

```
System.out.println("Protocol: "+url.getProtocol());

System.out.println("Host Name: "+url.getHost());

System.out.println("Port Number: "+url.getPort());

System.out.println("Default Port Number: "+url.getDefaultPort());

System.out.println("Query String: "+url.getQuery());

System.out.println("Path: "+url.getPath());

System.out.println("File: "+url.getFile());


}catch(Exception e){System.out.println(e);}

}

}
```

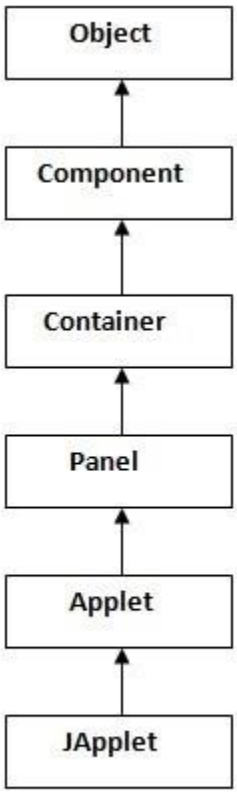
Output

```
Protocol: https

Host Name: www.google.com

Port Number: -1

Default Port Number: 443

Query String: q=javatpoint&oq=javatpoint&sourceid=chrome&ie=UTF-8

Path: /search

File: /search?q=javatpoint&oq=javatpoint&sourceid=chrome&ie=UTF-8
```

Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Hierarchy of Applet



Lifecycle of Java Applet

- Applet is initialized.
- Applet is started.
- Applet is painted.
- Applet is stopped.
- Applet is destroyed.

Lifecycle methods for Applet

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

public void init(): is used to initialized the Applet. It is invoked only once.

public void start(): is invoked after the init() method or browser is maximized. It is used to start the Applet.

public void stop(): is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.

public void destroy(): is used to destroy the Applet. It is invoked only once.

java.awt.Component class

The Component class provides 1 life cycle method of applet.

public void paint(Graphics g): is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

Simple example of Applet by html file

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

//First.java

```
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{
```

```
public void paint(Graphics g){
g.drawString("welcome",150,150);
}
}
```

myapplet.html

```
<html>
<body>
<applet code="First.class" width="300" height="300">
</applet>
</body>
</html>
```

Displaying Graphics in Applet

java.awt.Graphics class provides many methods for graphics programming.

Commonly used methods of Graphics class:

public abstract void drawString(String str, int x, int y): is used to draw the specified string.

public void drawRect(int x, int y, int width, int height): draws a rectangle with the specified width and height.

public abstract void fillRect(int x, int y, int width, int height): is used to fill rectangle with the default color and specified width and height.

public abstract void drawOval(int x, int y, int width, int height): is used to draw oval with the specified width and height.

public abstract void fillOval(int x, int y, int width, int height): is used to fill oval with the default color and specified width and height.

public abstract void drawLine(int x1, int y1, int x2, int y2): is used to draw line between the points(x1, y1) and (x2, y2).

public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer): is used draw the specified image.

public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle): is used draw a circular or elliptical arc.

public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle): is used to fill a circular or elliptical arc.

public abstract void setColor(Color c): is used to set the graphics current color to the specified color.

public abstract void setFont(Font font): is used to set the graphics current font to the specified font.

Example of Graphics in applet:

```
import java.applet.Applet;
import java.awt.*;
public class GraphicsDemo extends Applet{
```

```
public void paint(Graphics g){
g.setColor(Color.red);
g.drawString("Welcome",50, 50);
g.drawLine(20,30,20,300);
g.drawRect(70,100,30,30);
g.fillRect(170,100,30,30);
g.drawOval(70,200,30,30);

g.setColor(Color.pink);
g.fillOval(170,200,30,30);
g.drawArc(90,150,30,30,30,270);
g.fillArc(270,150,30,30,0,180);
}
}
```

myapplet.html

```
<html>
<body>
<applet code="GraphicsDemo.class" width="300" height="300">
</applet>
</body>
</html>
```

Displaying Image 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.

Syntax of drawImage() method:

public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer): is used draw the specified image.

How to get the object of Image:

The java.applet.Applet class provides getImage() method that returns the object of Image.

Syntax:

public Image getImage(URL u, String image){}

Other required methods of Applet class to display image:

public URL getDocumentBase(): is used to return the URL of the document in which applet is embedded.

public URL getCodeBase(): is used to return the base URL.

Example of displaying image in applet:

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

```
public class DisplayImage extends Applet {
  Image picture;
  public void init() {
    picture = getImage(getDocumentBase(),"sonoo.jpg");
  }
  public void paint(Graphics g) {
    g.drawImage(picture, 30,30, this);
  }
  }
```

In the above example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method of is ImageObserver object. The Component class implements ImageObserver interface. So current class object would also be treated as ImageObserver because Applet class indirectly extends the Component class.

myapplet.html

```
<html>
<body>
<applet code="DisplayImage.class" width="300" height="300">
</applet>
</body>
</html>
```

EventHandling in Applet

As we perform event handling in AWT or Swing, we can perform it in applet also. Let's see the simple example of event handling in applet that prints a message by click on the button.

Example of EventHandling in applet:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class EventApplet extends Applet implements ActionListener{
Button b;
TextField tf;

public void init(){
tf=new TextField();
tf.setBounds(30,40,150,20);

b=new Button("Click");
b.setBounds(80,150,60,50);

add(b);add(tf);
b.addActionListener(this);
```

```
setLayout(null);
}

 public void actionPerformed(ActionEvent e){
  tf.setText("Welcome");
 }
}
```

In the above example, we have created all the controls in init() method because it is invoked only once.

myapplet.html

```
<html>
<body>
<applet code="EventApplet.class" width="300" height="300">
</applet>
</body>
</html>
```

Digital clock in Applet

Digital clock can be created by using the Calendar and SimpleDateFormat class. Let's see the simple example:

Example of Digital clock in Applet:

```
import java.applet.*;
import java.awt.*;
import java.util.*;
import java.text.*;

public class DigitalClock extends Applet implements Runnable {

  Thread t = null;
  int hours=0, minutes=0, seconds=0;
  String timeString = "";

  public void init() {
    setBackground( Color.green);
  }

  public void start() {
     t = new Thread( this );
```

```
    t.start();
}


public void run() {
  try {
    while (true) {

      Calendar cal = Calendar.getInstance();
      hours = cal.get( Calendar.HOUR_OF_DAY );
      if ( hours > 12 ) hours -= 12;
      minutes = cal.get( Calendar.MINUTE );
      seconds = cal.get( Calendar.SECOND );

      SimpleDateFormat formatter = new SimpleDateFormat("hh:mm:ss");
      Date date = cal.getTime();
      timeString = formatter.format( date );

      repaint();
      t.sleep( 1000 );  // interval given in milliseconds
    }
  }
  catch (Exception e) { }
```

```
  }


 public void paint( Graphics g ) {
    g.setColor( Color.blue );
    g.drawString( timeString, 50, 50 );
  }
}
```

In the above example, getX() and getY() method of MouseEvent is used to get the current x-axis and y-axis. The getGraphics() method of Component class returns the object of Graphics.

myapplet.html

```
<html>
<body>
<applet code="DigitalClock.class" width="300" height="300">
</applet>
</body>
</html>
```

Analog clock in Applet

Analog clock can be created by using the Math class. Let's see the simple example:

Example of Analog clock in Applet:

```
import java.applet.*;
import java.awt.*;
import java.util.*;
import java.text.*;

public class MyClock extends Applet implements Runnable {

  int width, height;
  Thread t = null;
  boolean threadSuspended;
  int hours=0, minutes=0, seconds=0;
  String timeString = "";

  public void init() {
    width = getSize().width;
    height = getSize().height;
    setBackground( Color.black );
  }
```

```
public void start() {
  if ( t == null ) {
    t = new Thread( this );
    t.setPriority( Thread.MIN_PRIORITY );
    threadSuspended = false;
    t.start();
  }
  else {
    if ( threadSuspended ) {
      threadSuspended = false;
      synchronized( this ) {
        notify();
      }
    }
  }
}

public void stop() {
  threadSuspended = true;
}

public void run() {
```

```
try {
  while (true) {

    Calendar cal = Calendar.getInstance();
    hours = cal.get( Calendar.HOUR_OF_DAY );
    if ( hours > 12 ) hours -= 12;
    minutes = cal.get( Calendar.MINUTE );
    seconds = cal.get( Calendar.SECOND );

    SimpleDateFormat formatter
      = new SimpleDateFormat( "hh:mm:ss", Locale.getDefault() );
    Date date = cal.getTime();
    timeString = formatter.format( date );

    // Now the thread checks to see if it should suspend itself
    if ( threadSuspended ) {
      synchronized( this ) {
        while ( threadSuspended ) {
          wait();
        }
      }
    }
    repaint();
```

```
        t.sleep( 1000 );  // interval specified in milliseconds
      }
    }
    catch (Exception e) { }
}

void drawHand( double angle, int radius, Graphics g ) {
    angle -= 0.5 * Math.PI;
    int x = (int)( radius*Math.cos(angle) );
    int y = (int)( radius*Math.sin(angle) );
    g.drawLine( width/2, height/2, width/2 + x, height/2 + y );
}

void drawWedge( double angle, int radius, Graphics g ) {
    angle -= 0.5 * Math.PI;
    int x = (int)( radius*Math.cos(angle) );
    int y = (int)( radius*Math.sin(angle) );
    angle += 2*Math.PI/3;
    int x2 = (int)( 5*Math.cos(angle) );
    int y2 = (int)( 5*Math.sin(angle) );
    angle += 2*Math.PI/3;
    int x3 = (int)( 5*Math.cos(angle) );
    int y3 = (int)( 5*Math.sin(angle) );
```

```
    g.drawLine( width/2+x2, height/2+y2, width/2 + x, height/2 + y );
    g.drawLine( width/2+x3, height/2+y3, width/2 + x, height/2 + y );
    g.drawLine( width/2+x2, height/2+y2, width/2 + x3, height/2 + y3 );
  }

  public void paint( Graphics g ) {
    g.setColor( Color.gray );
    drawWedge( 2*Math.PI * hours / 12, width/5, g );
    drawWedge( 2*Math.PI * minutes / 60, width/3, g );
    drawHand( 2*Math.PI * seconds / 60, width/2, g );
    g.setColor( Color.white );
    g.drawString( timeString, 10, height-10 );
  }
}
```

myapplet.html

```
<html>
<body>
<applet code="MyClock.class" width="300" height="300">
</applet>
</body>
</html>
```

RMI (Remote Method Invocation)

The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects stub and skeleton.

Understanding stub and skeleton

RMI uses stub and skeleton object for communication with the remote object

A remote object is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

stub

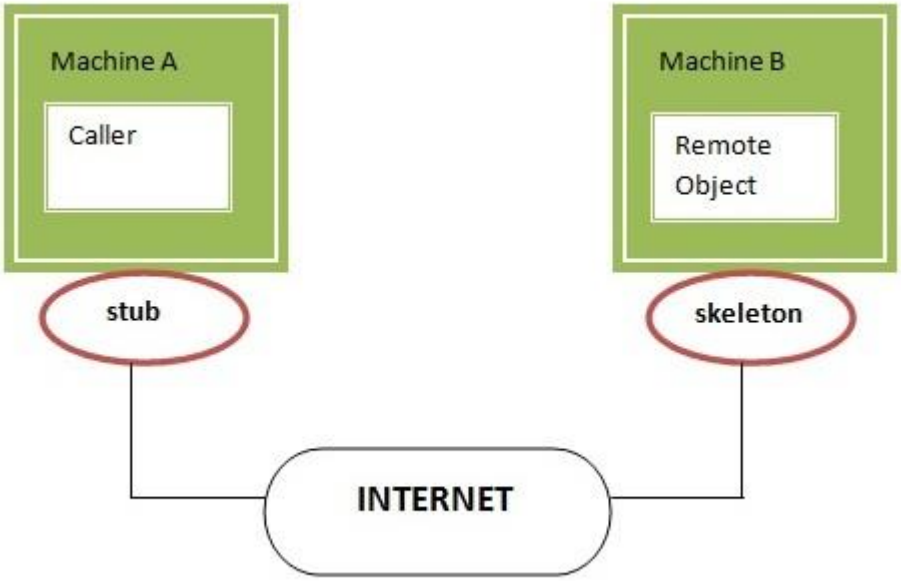
The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

- It initiates a connection with remote Virtual Machine (JVM),
- It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
- It waits for the result
- It reads (unmarshals) the return value or exception, and
- It finally, returns the value to the caller.

skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

- It reads the parameter for the remote method
- It invokes the method on the actual remote object, and
- It writes and transmits (marshals) the result to the caller.



Understanding requirements for the distributed applications

If any application performs these tasks, it can be distributed application.

- The application need to locate the remote method
- It need to provide the communication with the remote objects, and
- The application need to load the class definitions for the objects.
- The RMI application have all these features, so it is called the distributed application.

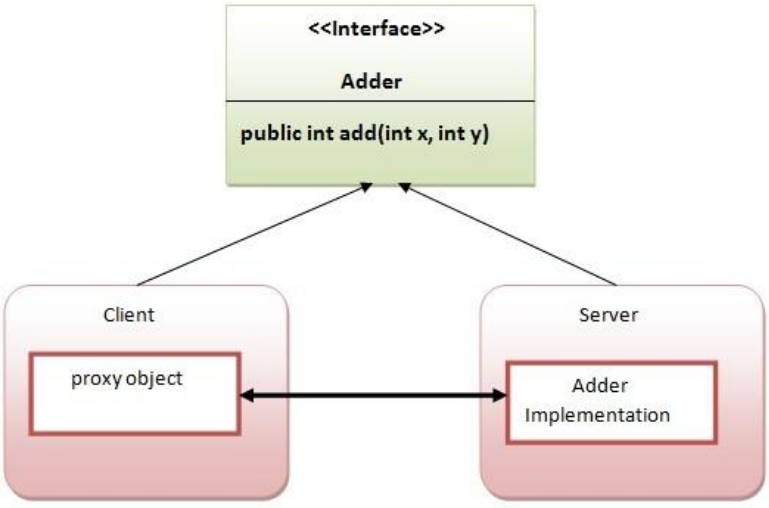
Java RMI Example

The is given the 6 steps to write the RMI program.

- Create the remote interface
- Provide the implementation of the remote interface
- Compile the implementation class and create the stub and skeleton objects using the rmic tool
- Start the registry service by rmiregistry tool
- Create and start the remote application
- Create and start the client application

RMI Example

In this example, we have followed all the 6 steps to create and run the rmi application. The client application need only two files, remote interface and client application. In the rmi application, both client and server interacts with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.



1) create the remote interface

For creating the remote interface, extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, we are creating a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException.

```
import java.rmi.*;
public interface Adder extends Remote{
public int add(int x,int y)throws RemoteException;
}
```

2) Provide the implementation of the remote interface

Now provide the implementation of the remote interface. For providing the implementation of the Remote interface, we need to

Either extend the UnicastRemoteObject class,

or use the exportObject() method of the UnicastRemoteObject class

In case, you extend the UnicastRemoteObject class, you must define a constructor that declares RemoteException.

```
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Adder{
AdderRemote()throws RemoteException{
super();
}
```

```
public int add(int x,int y){return x+y;}

}
```

3) create the stub and skeleton objects using the rmic tool.

Next step is to create stub and skeleton objects using the rmi compiler. The rmic tool invokes the RMI compiler and creates stub and skeleton objects.

```
rmic AdderRemote
```

4) Start the registry service by the rmiregistry tool

Now start the registry service by using the rmiregistry tool. If you don't specify the port number, it uses a default port number. In this example, we are using the port number 5000.

```
rmiregistry 5000
```

5) Create and run the server application

Now rmi services need to be hosted in a server process. The Naming class provides methods to get and store the remote object. The Naming class provides 5 methods.

public static java.rmi.Remote lookup(java.lang.String) throws java.rmi.NotBoundException, java.net.MalformedURLException, java.rmi.RemoteException;	It returns the reference of the remote object.
public static void bind(java.lang.String, java.rmi.Remote) throws java.rmi.AlreadyBoundException, java.net.MalformedURLException, java.rmi.RemoteException;	It binds the remote object with the given name.
public static void unbind(java.lang.String) throws java.rmi.RemoteException, java.rmi.NotBoundException, java.net.MalformedURLException;	It destroys the remote object which is bound with the given name.
public static void rebind(java.lang.String, java.rmi.Remote) throws java.rmi.RemoteException, java.net.MalformedURLException;	It binds the remote object to the new name.
public static java.lang.String[] list(java.lang.String) throws java.rmi.RemoteException, java.net.MalformedURLException;	It returns an array of the names of the remote objects bound in the registry.

In this example, we are binding the remote object by the name sonoo.

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
public static void main(String args[]){
try{
Adder stub=new AdderRemote();
Naming.rebind("rmi://localhost:5000/sonoo",stub);
}catch(Exception e){System.out.println(e);}
}
}
```

6) Create and run the client application

At the client we are getting the stub object by the lookup() method of the Naming class and invoking the method on this object. In this example, we are running the server and client applications, in the same machine so we are using localhost. If you want to access the remote

object from another machine, change the localhost to the host name (or IP address) where the remote object is located.

```
import java.rmi.*;
public class MyClient{
public static void main(String args[]){
try{
Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
System.out.println(stub.add(34,4));
}catch(Exception e){}
}
}
```