Networking

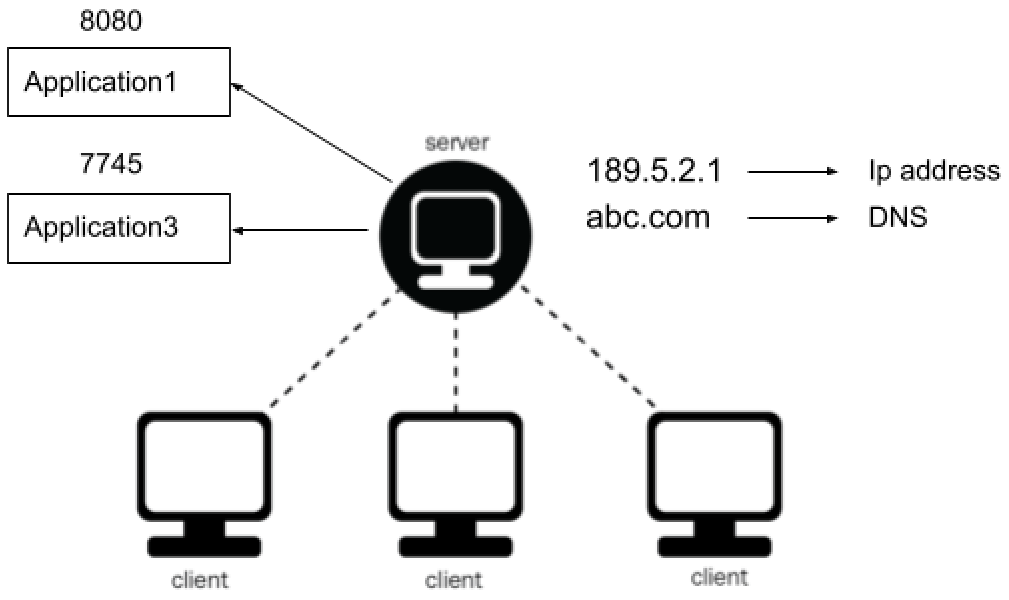
Basics of Networking

IP Address:

An IP address is a unique address that identifies a device on the internet or a local network. IP stands for "Internet Protocol," which is the set of rules governing the format of data sent via the internet or local network.

DNS

The Domain Name System (DNS) is the phonebook of the Internet. Humans access information online through domain names, like nytimes.com or espn.com. Web browsers interact through Internet Protocol (IP) addresses. DNS translates domain names to IP addresses so browsers can load Internet resources.



abc.com - > 189.5.2.1:8080

When client said abc.com it means 189.5.2.1:8080

Socket

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:

1. IP Address of Server
2. Port number.

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

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

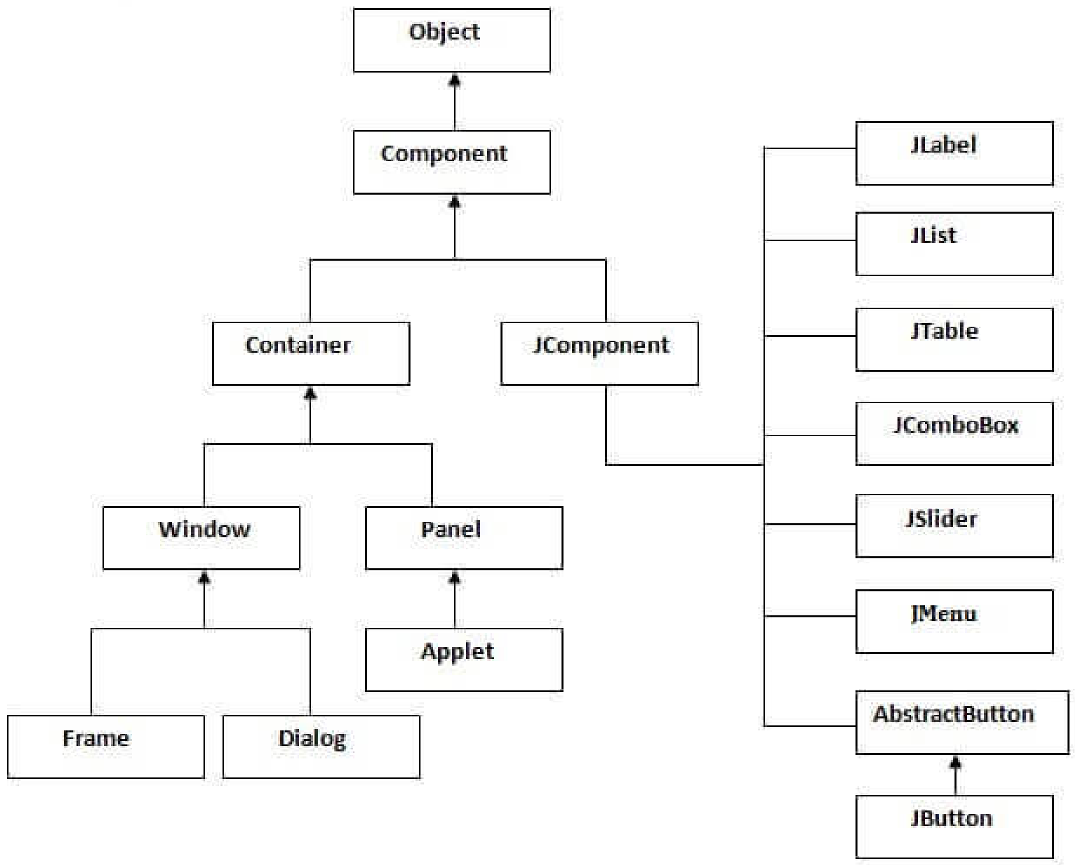
Method	Description
1) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket.

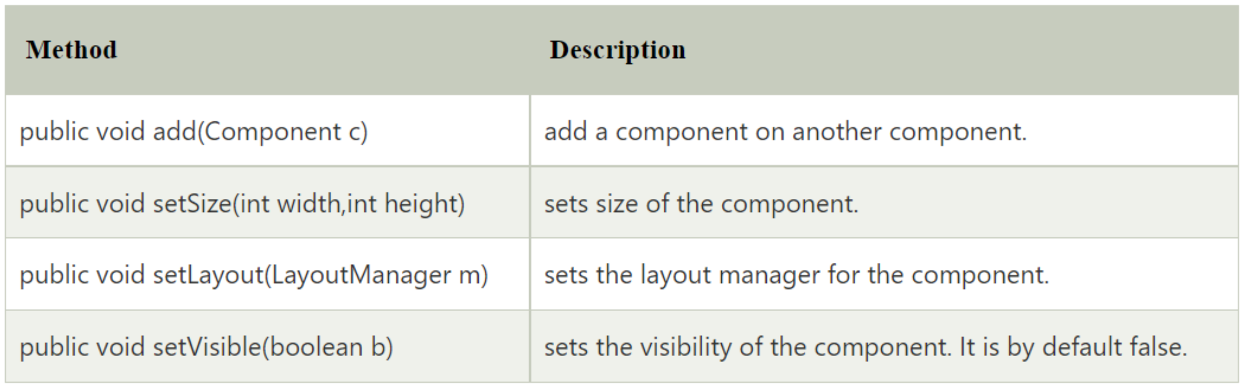
Swing

Java Swing tutorial is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

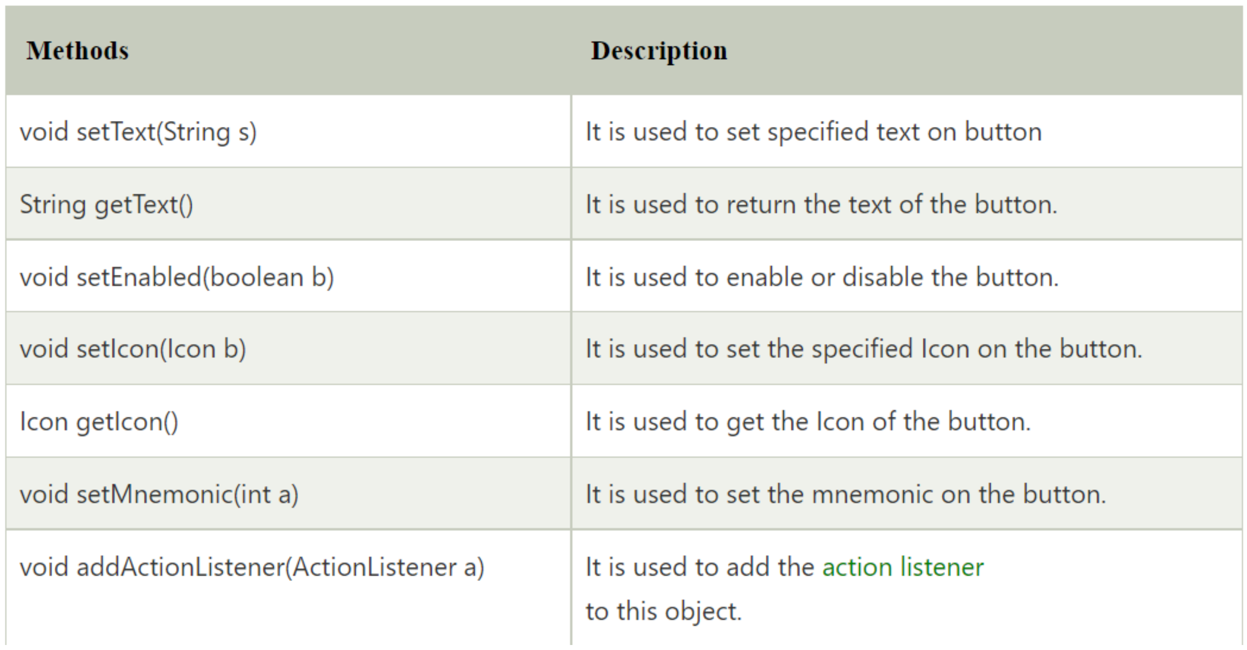




Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

JButton

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.



Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

JLabel

The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.

Methods	Description
String getText()	t returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.
void setHorizontalAlignment(int alignment)	It sets the alignment of the label's contents along the X axis.
Icon getIcon()	It returns the graphic image that the label displays.
int getHorizontalAlignment()	It returns the alignment of the label's contents along the X axis.

JTextField

The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

Methods	Description
void addActionListener(ActionListener l)	It is used to add the specified action listener to receive action events from this textfield.
Action getAction()	It returns the currently set Action for this ActionEvent source, or null if no Action is set.
void setFont(Font f)	It is used to set the current font.
void removeActionListener(ActionListener l)	It is used to remove the specified action listener so that it no longer receives action events from this textfield.

Default Method

Java provides a facility to create default methods inside the interface. Methods which are defined inside the interface and tagged with default are known as default methods. These methods are non-abstract methods.

```
interface Sayable{

    default void say(){
        System.out.println("Hello, this is default method");
    }

    void sayMore(String msg);
}
public class DefaultMethods implements Sayable{
    public void sayMore(String msg){
        System.out.println(msg);
    }
    public static void main(String[] args) {
        DefaultMethods dm = new DefaultMethods();
        dm.say();
        dm.sayMore("this is definition of abstract method");

    }
}
```

Functional Interface

An Interface that contains exactly one abstract method is known as functional interface. It can have any number of default, static methods but can contain only one abstract method. It can also declare methods of object class.

Functional Interface is also known as Single Abstract Method Interfaces or SAM Interfaces. It is a new feature in Java, which helps to achieve functional programming approach.

```
@FunctionalInterface
interface sayable{
   void say(String msg);
}
```

Lambda expression ->

Lambda expression is a new and important feature of Java which was included in Java SE 8. It provides a clear and concise way to represent one method interface using an expression. It is very useful in the collection library. It helps to iterate, filter and extract data from collection.

The Lambda expression is used to provide the implementation of an interface which has a functional interface. It saves a lot of code. In case of lambda expression, we don't need to define the method again for providing the implementation. Here, we just write the implementation code.

Java lambda expression is treated as a function, so the compiler does not create a .class file.

No Parameter Syntax

```
() -> {
//Body of no parameter lambda
}
```

One Parameter Syntax

```
(p1) -> {
//Body of single parameter lambda
}
```

Two Parameter Syntax

```
(p1,p2) -> {
//Body of multiple parameter lambda
}
```

```
@FunctionalInterface
interface Drawable{
   public void draw();
}

public class LambdaExpressionExample {
   public static void main(String[] args) {
      int width=10;

      Drawable d2=()->{
         System.out.println("Drawing "+width);
      };
      d2.draw();
   }
}
```