Polymorphism using Methods

If the same method performs different tasks, then that method is said to exhibit polymorphism. This statement is tricky. Let us stop and think again, how is it possible for a method to perform different tasks? It is possible only when the method assumes different bodies. It is something like this, take two persons with the same name 'Ravi'. Because the name is same but the persons are different, they can perform different tasks. In the same way, there may be two methods with the same name and they can perform different tasks. When we call the methods, we use same name but the task will be different depending on which method (body) is called.

Now, the crucial thing is to decide which method is called in a particular context. This decision may happen either at compile time or at runtime. This will lead to two types of polymorphism, Static polymorphism and Dynamic polymorphism. The words 'static' represents at compile time and 'dynamic' represents at run time. Let us discuss dynamic polymorphism first.

Dynamic Polymorphism

The polymorphism exhibited at runtime is called dynamic polymorphism. This means when a method is called, the method call is bound to the method body at the time of running the program, dynamically. In this case, Java compiler does not know which method is called at the time of compilation. Only JVM knows at runtime which method is to be executed. Hence, this is also called 'runtime polymorphism' or 'dynamic binding'.

Let us take a class 'Sample' with two instance methods having same name as:

```
void add(int a, int b)
{
        System.out.println("Sum of two= "+ (a+b));
}
void add(int a, int b, int c)
{
        System.out.println("Sum of three= "+ (a+b+c));
}
```

The bodies of these methods are different and hence they can perform different tasks. For example, the first method adds two integer numbers and the second one adds three integer numbers. However, to call these methods, we use the same method name. Suppose, we called the method as:

❑ `s.add(10, 15);` `//s` is the object of Sample class.

Now, who will decide which method is to be executed? Is it Java compiler or JVM? Because the methods are called by using an object, the Java compiler can not decide at the time of compilation which method is actually called by the user. It has to wait till the object is created for Sample class. And the creation of object takes place at runtime by JVM. Now, JVM should decide which method is actually called by the user at runtime (dynamically).

The question is how JVM recognizes which method is called, when both the methods have same name. For this, JVM observes the signature of the methods. Method signature consists of a method name and its parameters. Even if, two methods have same name, their signature may vary. For example, two human beings may have same name but their signatures will differ.

Important Interview Question

What is method signature?

Method signature represents the method name along with method parameters.

When there is a difference in the method signatures, then the JVM understands both the methods are different and can call the appropriate method. The difference in the method signatures will arise because of one of the following reasons:

- There may be a difference in the number of parameters passed to the methods.

For example,

```
void add(int a, int b)
void add(int a, int b, int c)
```

In this case, if we call add(10, 15) then JVM executes the first method and if we call add(10, 15, 22) then it runs the second method.

- Or, there may be a difference in the data types of parameters.

For example,

```
void add(int a, float b)
void add(double a, double b)
```

In this case, if we call the method add(10, 5.5f) then JVM goes for the first method and if it is add(10.5, 22.9) then JVM runs the second one.

- Or, there may be a difference in the sequence (orderliness) of the parameters.

For example,

```
void add(int a, float b)
void add(float a, int b)
```

In this case, if we call add(10, 5.55) then JVM executes the first method and if it is add(5.5, 20) then it goes for the second method.

JVM matches the values passed to the method at the time of method call with the method signature and picks up the appropriate method. In this way, difference in the method signatures helps JVM to identify the correct method and execute it. Please see Program 1 to understand this.

Program 1: Write a program to create Sample class which contains two methods with the same name but with different signatures.

```
//Dynamic polymorphism
class Sample
{
        //method to add two values
        void add(int a,int b)
        {
                System.out.println("Sum of two= "+ (a+b));
        }
        //method to add three values
        void add(int a, int b, int c)
        {
                System.out.println("Sum of three= "+ (a+b+c));
        }
}
class Poly
{
        public static void main(String args[ ])
        {
                //create Sample class object
                Sample s = new Sample();
                //call add() and pass two values
                s.add(10,15);    //This call is bound with first method
                //call add() and pass three values
                s.add(10,15,20);  //This call is bound with second method
        }
}
```

25	02798	D/052/21	ANURAG SINHA

correction. These frames are then sent to Data link layer which dispatches them to correct destination computer on the network. The last layer, which is called the Physical layer, is used to physically transmit data on the network using the appropriate hardware. See Figure 25.1.

Of course, to send data from one place to another, first of all the computers should be correctly identified on the network. This is done with the help of IP addresses. An IP address is a unique identification number given to every computer on the network. It contains four integer numbers in the range of 0 to 255 and separated by a dot as:

```
87.248.113.14
```

This IP address may represent, for example a website on a server machine on Internet as:

www.yahoo.com

Therefore, to open 'yahoo.com' site, we can type the site address as 'www.yahoo.com' or its IP address as '87.248.113.14'. But when we type the IP address in numeric form, that number is mapped to the website automatically. This mapping service is available on Internet, which is called 'DNS' (Domain Naming service).

Important Interview Question

What is IP address?

An IP address is a unique identification number allotted to every computer on a network or Internet. IP address contains some bytes which identify the network and the actual computer inside the network.

What is DNS?

Domain Naming Service is a service on Internet that maps the IP addresses with corresponding website names.

On Internet, IP addresses of 4 bytes are used and this version is called IP address version 4. The next new version of IP address is version 6, which uses 16 bytes to identify a computer.

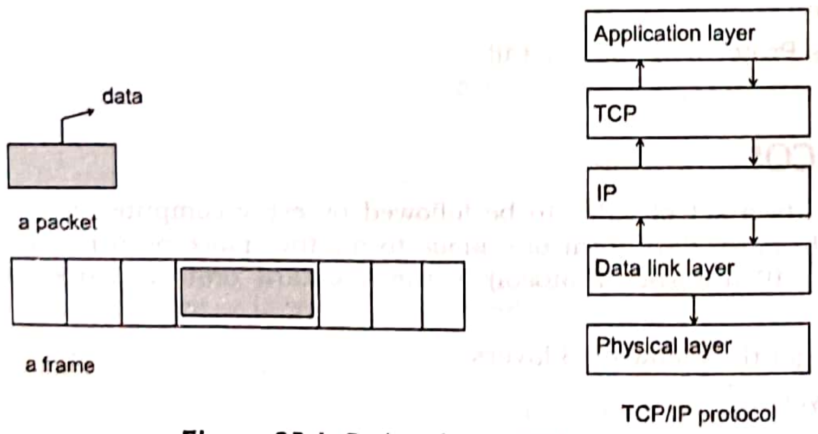


Figure 25.1 Packet, frame, TCP/IP layers

TCP/IP takes care of number of bits sent and whether all the bits are received duly by the destination computer. So it is called 'connection oriented reliable protocol'. Every transmitted bit is accountable in this protocol. Hence, this protocol is highly suitable for transporting data reliably on a network. Almost all the protocols on Internet use TCP/IP model internally.

HTTP (hyper text transfer protocol) is the most widely used protocol on Internet, which is used to transfer web pages (.html files) from one computer to another computer on Internet. FTP (file transfer protocol) is useful to download or upload files from and to the server. SMTP (simple mail transfer protocol) is useful to send mails on network. POP (post office protocol) is useful to receive mails into the mail boxes.

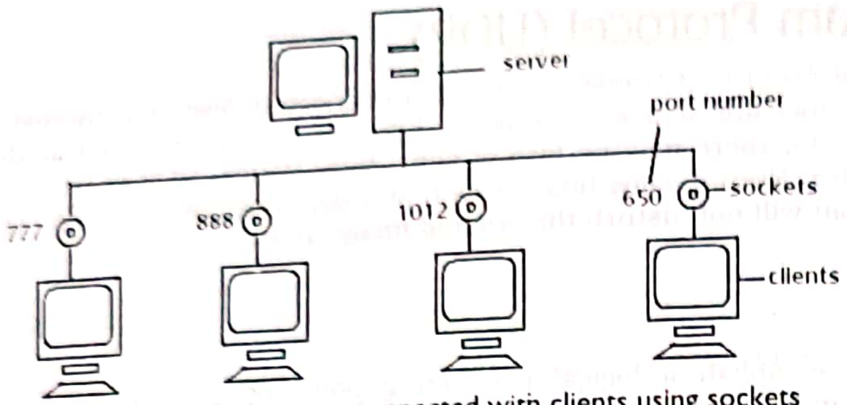


Figure 25.2 A server connected with clients using sockets

A socket, at server side is called 'server socket ' and is created using `ServerSocket` class in Java. A socket, at client side is called 'Socket' and is created using `Socket` class. Both the `ServerSocket` and `Socket` classes are available in `java.net` package. Of course, a server socket may not be necessarily at server side; it may be created at client side also, if the client acts as server. Similarly, a client socket may also exist at server side, if the server acts as client.

Knowing IP Address

It is possible to know the IP Address of a website on Internet with the help of `getByName()` method of `InetAddress` class of `java.net` package. The `getByName()` method takes host name (server name) and returns `InetAddress`, which is nothing but the IPAddress of that server. See the following program.

Program 1: Write a program to accept a website name and return its IPAddress, after checking it on Internet.

Note

This program should be executed on a system which is connected to Internet.

```
//Knowing IPAddress of a website
import java.io.*;
import java.net.*;
class Address
{
     public static void main(String args[ ]) throws IOException
     {
            //accept name of website from keyboard
            BufferedReader br = new BufferedReader(new
             InputStreamReader(System.in));
            System.out.print("Enter a website name: ");
            String site = br.readLine();
            try{
                   //getByName() method accepts site name and returns its IP
                   //Address
                   InetAddress ip = InetAddress.getByName(site);
                   System.out.println("The IP Address is: "+ ip);
            }catch(UnknownHostException ue)
            {
                   System.out.println("Website not found");
            }
     }
}
```

Output:

```
C:\> javac Address.java
C:\> java Address
Enter a website name: www.yahoo.com
The IP Address is: www.yahoo.com/87.248.113.14
```

URL

URL (Uniform Resource Locator) represents the address that is specified to access some information or resource on Internet. Look at the example URL:

`http://www.dreamtechpress.com:80/index.html`

The URL contains 4 parts:

- The protocol to use (`http://`).
- The server name or IP address of the server (`www.dreamtechpress.com`).
- The third part represents port number, which is optional (`:80`).
- The last part is the file that is referred. This would be generally `index.html` or `home.html` file (`/index.html`).

URL is represented by a class 'URL' in `java.net` package. To create an object to URL, we can use the following formats:

```
    URL obj = new URL(String protocol, String host, int port, String path);
```

Or,

```
    URL obj = new URL(String protocol, String host, String path);
```

The following program accesses the different parts of the URL supplied to URL object and displays them.

Program 2: Write a program to retrieve different parts of a URL supplied to URL class object.

```
//URL
import java.net.*;
class MyURL
{
        public static void main(String args[ ]) throws Exception
        {
                URL obj = new URL("http://dreamtechpress.com/index.html");
                System.out.println("Protocol: "+ obj.getProtocol());
                System.out.println("Host: "+ obj.getHost());
                System.out.println("File: "+ obj.getFile());
                System.out.println("Port: "+ obj.getPort());
                System.out.println("Path: "+ obj.getPath());
                System.out.println("External form: "+ obj.toExternalForm());
        }
}
```

Output:

```
C:\> javac MyURL.java
C:\> java MyURL
Protocol: http
Host: dreamtechpress.com
File: /index.html
```

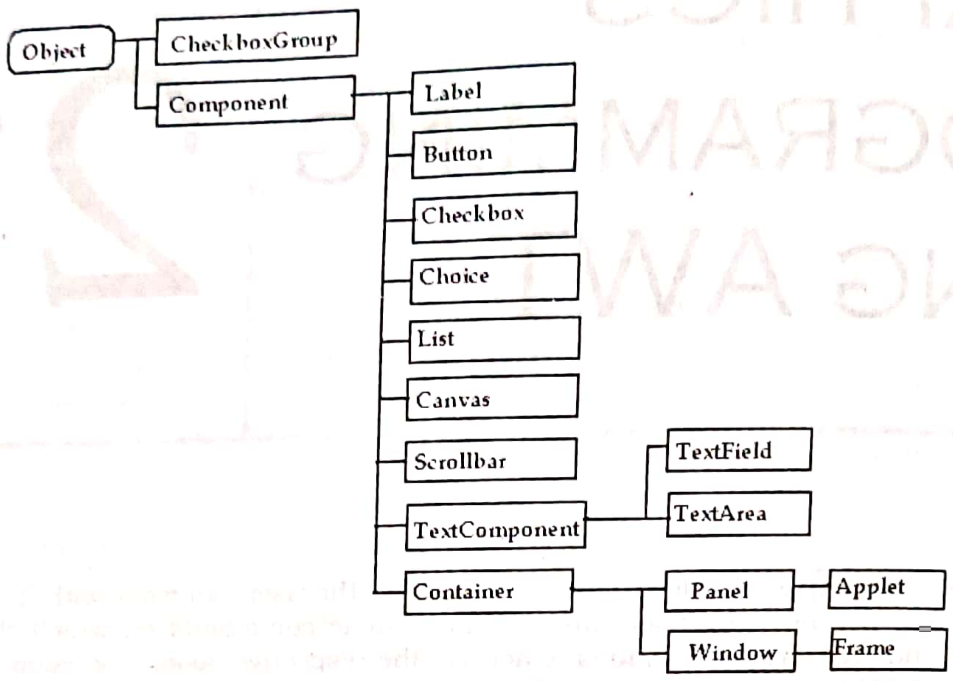


Figure 27.1 Classes of AWT

Components

A Component represents an object which is displayed pictorially on the screen. For example, we create an object of `Button` class as:

```
Button b = new Button();
```

Now, `b` is the object of `Button` class. If we display this `b` on the screen, it displays a push button. Therefore, the object `b`, on going to the screen is becoming a component called 'Push button'. In the same way, any component is a graphical representation of an object. Push buttons, radio buttons, check boxes, etc. are all components.

Window and Frame

A window represents any imaginary rectangular area on the screen without any borders or title bar. A frame represents a window with some title and borders. See Figure 27.2. In any application, we create frames to represent various screens like input screens where the user can type some data for the application and output screens where the result may be displayed in a particular form. Such screens are nothing but frames only.

Important Interview Question

What is the difference between a window and a frame?

A window is a frame without any borders and title, whereas a frame contains borders and title.

Closing the Frame

We know Frame is also a component. We want to close the frame by clicking on its close button. Let us follow these steps to see how to use event delegation model to do this:

- We should attach a listener to the frame component. Remember, all listeners are available in `java.awt.event` package. The most suitable listener to the frame is 'window listener'. It can be attached using `addWindowListener()` method as:

```
f.addWindowListener(WindowListener obj);
```

Please note that the `addWindowListener()` method has a parameter that is expecting object of `WindowListener` interface. Since it is not possible to create an object to an interface, we should create an object to the implementation class of the interface and pass it to the method.

- Implement all the methods of the `WindowListener` interface. The following methods are found in `WindowListener` interface:

```
public void windowActivated(WindowEvent e)
public void windowClosed(WindowEvent e)
public void windowClosing(WindowEvent e)
public void windowDeactivated(WindowEvent e)
public void windowDeiconified(WindowEvent e)
public void windowIconified(WindowEvent e)
public void windowOpened(WindowEvent e)
```

In all the preceding methods, `WindowListener` interface calls the `public void windowClosing()` method when the frame is being closed. So, implementing this method alone is enough, as:

```
public void windowClosing(WindowEvent e)
{
        //close the application
        System.exit(0);
}
```

For the remaining methods, we can provide empty body.

- So, when the frame is closed, the body of this method is executed and the application gets closed. In this way, we can handle the frame closing event.

These steps are shown in Program 3.

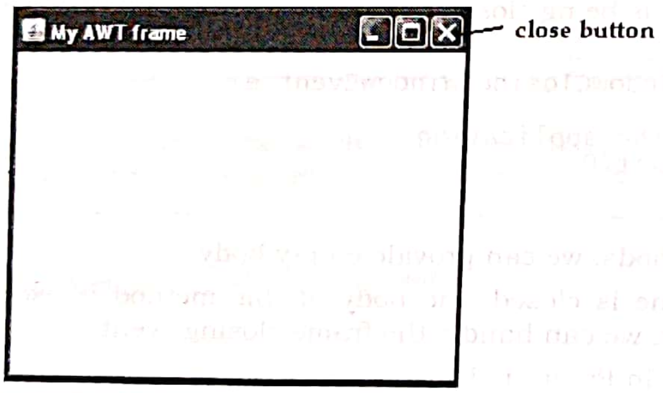
Program 3: Write a program which first creates a frame and then closes it on clicking the close button.

```
//Creating a frame and closing it.
import java.awt.*;
import java.awt.event.*;
class MyFrame extends Frame
{
        public static void main(String args[])
        {
                //create a frame with title
                MyFrame f = new MyFrame();

                //set a title for the frame
                f.setTitle("My AWT frame");

                //set the size of the frame
                f.setSize(300,250);
```

```
                    //display the frame
                    f.setVisible(true);

                    //close the frame
                    f.addWindowListener(new Myclass());

            }
    }
    class Myclass implements WindowListener
    {
            public void windowActivated(WindowEvent e){}
            public void windowClosed(WindowEvent e){}
            public void windowClosing(WindowEvent e)
            {
                    System.exit(0);
            }
            public void windowDeactivated(WindowEvent e){}
            public void windowDeiconified(WindowEvent e){}
            public void windowIconified(WindowEvent e){}
            public void windowOpened(WindowEvent e){}

    }
```

Output

```
C:\> javac MyFrame.java
C:\> java MyFrame
CLICK ON CLOSE BUTTON, THE FRAME CLOSES
```



In this program, we not only create a frame but also close the frame when the user clicks on the close button. For this purpose, we use `WindowListener` interface.

Here, we had to mention all the methods of `WindowListener` interface, just for the sake of one method. This is really cumbersome. There is another way to escape this. There is a class `WindowAdapter` in `java.awt.event` package, that contains all the methods of the `WindowListener` interface with an empty implementation (body). If we can extend `Myclass` from this `WindowAdapter` class, then we need not write all the methods with empty implementation. We can write only that method which interests us. This is shown in Program 4.

Program 4: Write a program to close the frame using `WindowAdapter` class.

```
    //Creating a frame and closing it.
    import java.awt.*;
    import java.awt.event.*;
    class MyFrame extends Frame
    {

            public static void main(String args[])
    {
            //create a frame with title
            MyFrame f = new MyFrame();

            //set a title for the frame
            f.setTitle("My AWT frame");

            //set the size of the frame
            f.setSize(300,250);

            //display the frame
            f.setVisible(true);

            //close the frame
            f.addWindowListener(new MyClass());
    }
}
class MyClass extends WindowAdapter
{
        public void windowClosing(WindowEvent e)
        {
                System.exit(0);
        }
}
```

Output

```
C:\> javac MyFrame.java
C:\> java MyFrame
CLICK ON CLOSE BUTTON, THE FRAME CLOSES
```

Important Interview Question

What is an adapter class?

An adapter class is an implementation class of a listener interface which contains all methods implemented with empty body. For example, WindowAdapter is an adapter class of WindowListener interface. Adapter classes reduce overhead on programming while working with listener interfaces.

Please observe Program 4. In this, even the code of `Myclass` can be copied directly into `addWindowListener()` method, as:

```
    f.addWindowListener(new WindowAdapter()
    {
            public void windowClosing(WindowEvent e)
            {
                    System.exit(0);
            }
    });
```

This looks a bit confusing, but it is correct. We copy the code of `Myclass` into the method of `MyFrame` class. But, in the preceding code, we cannot find the name of `Myclass` anywhere in the code. It means the name of `Myclass` is hidden in `MyFrame` class and hence `Myclass` is an inner class in `MyFrame` class whose name is not mentioned. Such an inner class is called 'anonymous inner class'.

Important Interview Question

What is anonymous inner class?

Anonymous inner class is an inner class whose name is not mentioned, and for which only one object is created.