1. Basics of Networking

The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand.

The java.net package provides support for the two common network protocols –

- **TCP** – TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.
- **UDP** – UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

This chapter gives a good understanding on the following two subjects –

- **Socket Programming** – This is the most widely used concept in Networking and it has been explained in very detail.
- **URL Processing** – This would be covered separately. Click here to learn about URL Processing in Java language.
- **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.

Proxy Server

Proxy server is an intermediary server between client and the internet.

Proxy servers offers the following basic functionalities:

- Firewall and network data filtering.
- Network connection sharing
- Data caching

Proxy servers allow hiding, concealing and making your network id anonymous by hiding your IP address.

Purpose of Proxy Servers

Following are the reasons to use proxy servers:

- Monitoring and Filtering
- Improving performance
- Translation
- Accessing services anonymously
- Security

2. 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.

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.

Example of Java Socket Programming

Let's see a simple of java socket programming in which client sends a text and server receives 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);}
}
```

```
}
```

To execute this program open two command prompts and execute each program at each command prompt.
After running the client application, a message will be displayed on the server console.

Java DataOutputStream Class

Java DataOutputStream class allows an application to write primitive Java data types to the output stream in a machine-independent way.

Java application generally uses the data output stream to write data that can later be read by a data input stream.

Java DataOutputStream class methods

Method	Description
int size()	It is used to return the number of bytes written to the data output stream.
void write(int b)	It is used to write the specified byte to the underlying output stream.
void writeUTF(String str)	It is used to write a string to the output stream using UTF-8 encoding in portable manner.
void flush()	It is used to flushes the data output stream.

Java DataInputStream Class

Java DataInputStream class allows an application to read primitive data from the input stream in a machine-independent way.

Java application generally uses the data output stream to write data that can later be read by a data input stream.

Java DataInputStream class Methods

Method	Description
String readUTF()	It is used to read a string that has been encoded using the UTF-8 format.

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();
}}
```

3. 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:

http://www.javatpoint.com/java-tutorial 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.

Commonly used methods of Java URL class

The java.net.URL class provides many methods. The important methods of URL class are given below.

Method	Description
public String getProtocol()	It Returns The Protocol Of The URL.
public String getHost()	It Returns The Host Name Of The URL.
public String getPort()	It Returns The Port Number Of The URL.

public String getFile()	It Returns The File Name Of The URL.
public URLConnection openConnection()	It Returns The Instance Of URLConnection I.E. Associated With This URL

Example of Java URL class

```
//URLDemo.java
import java.io.*;
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
```

Java URLConnection class

The Java URLConnection class represents a communication link between the URL and the application. This class can be used to read and write data to the specified resource referred by the URL.

How to get the object of URLConnection class

The **openConnection()** method of URL class **returns the object of URLConnection class. Syntax**:

public URLConnection openConnection()throws IOException{}

Displaying source code of a webpage by URLConnecton class
The URLConnection class provides many methods, we can **display all the data of a webpage** by using the **getInputStream() method**. The getInputStream() method returns all the data of the specified URL in the stream that can be read and displayed.

Example of Java URLConnection class

```
import java.io.*;
import java.net.*;
public class URLConnectionExample { public static void main(String[]
args){ try{
URL url=new URL("http://www.javatpoint.com/java-tutorial");
URLConnection urlcon=url.openConnection();
InputStream stream=urlcon.getInputStream();
int i; while((i=stream.read())!=-1){ System.out.print((char)i);
}
}catch(Exception e){System.out.println(e);}
}
}
```

Java HttpURLConnection class

The Java HttpURLConnection class is http specific URLConnection. It works for HTTP protocol only.
By the help of HttpURLConnection class, you can information of any HTTP URL such as header information, status code, response code etc.
The java.net.HttpURLConnection is subclass of URLConnection class.

How to get the object of HttpURLConnection class
The **openConnection()** method of URL class **returns the object of URLConnection class.**

Syntax:
public URLConnection openConnection()throws IOException{}
You can typecast it to HttpURLConnection type as given below.

```
URL url=new URL("http://www.javatpoint.com/java-tutorial");
HttpURLConnection huc=(HttpURLConnection)url.openConnection();
```

Java HttpURLConnecton Example

```
import java.io.*;
import java.net.*;
public class HttpURLConnectionDemo{ public static void main(String[]
args){ try{
URL url=new URL("http://www.javatpoint.com/java-tutorial");
HttpURLConnection huc=(HttpURLConnection)url.openConnection();
for(int i=1;i<=8;i++){
System.out.println(huc.getHeaderFieldKey(i)+" =
"+huc.getHeaderField(i));
}
huc.disconnect();
}catch(Exception e){System.out.println(e);}
}
}
```

Output:

```
Date = Wed, 10 Dec 2014 19:31:14 GMT
Set-Cookie = JSESSIONID=D70B87DBB832820CACA5998C90939D48;
Path=/ Content-Type = text/html
Cache-Control = max-age=2592000
Expires = Fri, 09 Jan 2015 19:31:14 GMT Vary = Accept-Encoding,User-
Agent Connection = close
Transfer-Encoding = chunked
```

4. Java InetAddress class

Java InetAddress class represents an IP address. The java.net.InetAddress class provides methods

to get the IP of any host name for example www.google.com. Commonly used methods of

InetAddress class

Method	Description
1. **public static InetAddress getLocalHost() throws UnknownHostException**	it returns the instance of InetAddress containing LocalHost IP and name. it returns the instance of InetAdddress containing local host name and address.
2. **public String getHostName()**	it returns the host name of the IP address.
3. **public String getHostAddress()**	it returns the IP address in string format.

Example of Java InetAddress class

Let's see a simple example of InetAddress class to get ip address of www.javatpoint.com website.

```
import java.io.*;
import java.net.*;
public class InetDemo{
public static void main(String[] args){
try{
InetAddress ip=InetAddress.getByName("www.javatpoint.com");
System.out.println("Host Name: "+ip.getHostName());
System.out.println("IP Address: "+ip.getHostAddress());
}catch(Exception e){System.out.println(e);}
}
}
```

Output:

Host Name: www.javatpoint.com

IP Address: 206.51.231.148

5. java.security Package

All the classes which are related for management of security related concerns in the java

program are provided under this package. The various classes are discussed below:

Sr. No	Class	Description

1 **Permission** The class is meant to handle concerns related to accessing of system resources. The class is declared as:
public abstract class Permission extends Object implements Guard, Serializable
The class methods are inherited from the Object class

2 **Policy** The instances of the class determine whether the code being executed by the JVM has required permission to access the security policies. The class is declared as:
public abstract class Policy extends Object
The class methods are inherited from the Object class

ServerSocket Class Methods

The **java.net.ServerSocket** class is used by server applications to obtain a port and listen for client requests.

The ServerSocket class has four constructors –

Sr.No.	Method & Description
1	**public ServerSocket(int port) throws IOException** Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application.
2	**public ServerSocket(int port, int backlog) throws IOException** Similar to the previous constructor, the backlog parameter specifies how many incoming clients to store in a wait queue.
3	**public ServerSocket(int port, int backlog, InetAddress address) throws IOException** Similar to the previous constructor, the InetAddress parameter specifies the local IP address to bind to. The InetAddress is used for servers that may have multiple IP addresses, allowing the server to specify which of its IP addresses to accept client requests on.
4	**public ServerSocket() throws IOException** Creates an unbound server socket. When using this constructor, use the bind() method when you are ready to bind the server socket.

If the ServerSocket constructor does not throw an exception, it means that your application has successfully bound to the specified port and is ready for client requests.

Following are some of the common methods of the ServerSocket class –

Sr.No.	Method & Description
1	**public int getLocalPort()** Returns the port that the server socket is listening on. This method is useful if you passed in 0 as the port number in a constructor and let the server find a port for you.

2	**public Socket accept() throws IOException** Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the setSoTimeout() method. Otherwise, this method blocks indefinitely.
3	**public void setSoTimeout(int timeout)** Sets the time-out value for how long the server socket waits for a client during the accept().
4	**public void bind(SocketAddress host, int backlog)** Binds the socket to the specified server and port in the SocketAddress object. Use this method if you have instantiated the ServerSocket using the no-argument constructor.

When the ServerSocket invokes accept(), the method does not return until a client connects. After a client does connect, the ServerSocket creates a new Socket on an unspecified port and returns a reference to this new Socket. A TCP connection now exists between the client and the server, and communication can begin.

Socket Class Methods

The **java.net.Socket** class represents the socket that both the client and the server use to communicate with each other. The client obtains a Socket object by instantiating one, whereas the server obtains a Socket object from the return value of the accept() method.

The Socket class has five constructors that a client uses to connect to a server –

Sr.No.	Method & Description
1	**public Socket(String host, int port) throws UnknownHostException, IOException.** This method attempts to connect to the specified server at the specified port. If this constructor does not throw an exception, the connection is successful and the client is connected to the server.
2	**public Socket(InetAddress host, int port) throws IOException** This method is identical to the previous constructor, except that the host is denoted by an InetAddress object.

3	**public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException.** Connects to the specified host and port, creating a socket on the local host at the specified address and port.
4	**public Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException.** This method is identical to the previous constructor, except that the host is denoted by an InetAddress object instead of a String.
5	**public Socket()** Creates an unconnected socket. Use the connect() method to connect this socket to a server.

When the Socket constructor returns, it does not simply instantiate a Socket object but it actually attempts to connect to the specified server and port.

Some methods of interest in the Socket class are listed here. Notice that both the client and the server have a Socket object, so these methods can be invoked by both the client and the server.

Sr.No.	Method & Description
1	**public void connect(SocketAddress host, int timeout) throws IOException** This method connects the socket to the specified host. This method is needed only when you instantiate the Socket using the no-argument constructor.
2	**public InetAddress getInetAddress()** This method returns the address of the other computer that this socket is connected to.
3	**public int getPort()** Returns the port the socket is bound to on the remote machine.
4	**public int getLocalPort()** Returns the port the socket is bound to on the local machine.
5	**public SocketAddress getRemoteSocketAddress()** Returns the address of the remote socket.

6	**public InputStream getInputStream() throws IOException** Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.
7	**public OutputStream getOutputStream() throws IOException** Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket.
8	**public void close() throws IOException** Closes the socket, which makes this Socket object no longer capable of connecting again to any server.

InetAddress Class Methods

This class represents an Internet Protocol (IP) address. Here are following usefull methods which you would need while doing socket programming –

Sr.No.	Method & Description
1	**static InetAddress getByAddress(byte[] addr)** Returns an InetAddress object given the raw IP address.
2	**static InetAddress getByAddress(String host, byte[] addr)** Creates an InetAddress based on the provided host name and IP address.
3	**static InetAddress getByName(String host)** Determines the IP address of a host, given the host's name.
4	**String getHostAddress()** Returns the IP address string in textual presentation.
5	**String getHostName()** Gets the host name for this IP address.
6	**static InetAddress InetAddress getLocalHost()** Returns the local host.
7	**String toString()** Converts this IP address to a String.

TCP/IP Client Sockets

TCP/IP sockets are used to implement reliable, bidirectional, persistent, point-to-point, stream-based connections between hosts on the Internet. A socket can be used to connect Java's I/O system to other programs that may reside either on the local machine or on any other machine on the Internet.
There are two kinds of TCP sockets in Java. One is for servers, and the other is for clients. The **ServerSocket** class is designed to be a "listener," which waits for clients to connect before doing anything. Thus, **ServerSocket** is for servers. The **Socket** class is for clients. It is designed to connect to server sockets and initiate protocol exchanges. Because client sockets are the most commonly used by Java applications, they are examined here.
The creation of a **Socket** object implicitly establishes a connection between the client and server. There are no methods or constructors that explicitly expose the details of establishing that connection. Here are two constructors used to create client sockets:

Socket(String *hostName*, int *port*) throws UnknownHostException, IOException	Creates a socket connected to the named host and port.
Socket(InetAddress *ipAddress*, int *port*) throws IOException	Creates a socket using a preexisting **InetAddress** object and a port.

Instance Methods

Socket defines several instance methods. For example, a **Socket** can be examined at any time for the address and port information associated with it, by use of the following methods:

Sr. No	Method name	Syntax	Description
1	getAddress	public synchronized InetAddress getAddress()	The IP address of the packet.
2	getData	public synchronized byte[] getData()	The packet data.
3	getLength	public synchronized int getLength()	The packet length.
4	getPort	public synchronized int getPort()	
5	setAddress	public synchronized void setAddress(InetAddress iaddr)	This method sets the destination address for this packet
6	setData	public synchronized void setData(byte[] ibuf)	This method sets the data for this packet
7	setPort	public synchronized void setPort(int iport)	This method sets the destination port number for this packet.

1. getAddress
public synchronized InetAddress getAddress()
Returns
The IP address of the packet.
Description
If this packet has been received, the method returns the address of the machine that sent it. If the packet is being sent, the method returns the destination address.
2. getData
public synchronized byte[] getData()
Returns
The packet data.
Description
This method returns the data buffer associated with this DatagramPacket object. This data is either the data being sent or the data that has been received.
3. getLength
public synchronized int getLength()
Returns
The packet length.
Description
This method returns the length of the message in the buffer associated with this DatagramPacket. This length is either the length of the data being sent or the length of the data that has been received.
4. getPort
public synchronized int getPort()
Returns
The port number of the packet.
Description
If this packet has been received, the method returns the port number of the machine that sent it. If the packet is being sent, the method returns the destination port number.
5. setAddress
public synchronized void setAddress(InetAddress iaddr)
Availability
New as of JDK 1.1
Parameters
iaddr
The destination address for the packet.
Description
This method sets the destination address for this packet. When the packet is sent using DatagramSocket.send(), it is sent to the specified address.
6. setData
public synchronized void setData(byte[] ibuf)
Availability
New as of JDK 1.1
Parameters
ibuf
The data buffer for the packet.

Description

This method sets the data for this packet. When the packet is sent using DatagramSocket.send(), the specified data is sent.

7. setLength

public synchronized void setLength(int ilength)

Availability

New as of JDK 1.1

Parameters

ilength

The number of bytes to send.

Description

This method sets the length of the data to be sent for this packet. When the packet is sent using DatagramSocket.send(), the specified amount of data is sent.

8. setPort

public synchronized void setPort(int iport)

Availability

New as of JDK 1.1

Parameters

iport

The port number for the packet.

Description

This method sets the destination port number for this packet. When the packet is sent using DatagramSocket.send(), it is sent to the specified port.

InetAddress getInetAddress( ) : Returns the InetAddress associated with the Socket object. It returns null if the socket is not connected.
int getPort( ) : Returns the remote port to which the invoking Socket object is connected. It returns 0 if the socket is not connected.
int getLocalPort( ) : Returns the local port to which the invoking Socket object is bound. It returns –1 if the socket is not bound.

You can gain access to the input and output streams associated with a **Socket** by use of the **getInputStream( )** and **getOuptutStream( )** methods, as shown here. Each can throw an **IOException** if the socket has been invalidated by a loss of connection. These streams are used exactly like the I/O streams described in Chapter 20 to send and receive data.

InputStream getInputStream( ) throws IOException : Returns the InputStream associated with the invoking socket.

OutputStream getOutputStream( )throws IOException : Returns the OutputStream associated with the invoking socket.

Several other methods are available, including **connect( ),** which allows you to specify a new connection; **isConnected( ),** which returns true if the socket is connected to a server; **isBound( ),** which returns true if the socket is bound to an address; and **isClosed( ),** which returns true if the socket is closed. To close

a socket, call **close( )**. Closing a socket also closes the I/O streams associated with the socket. Beginning with JDK 7, **Socket** also implements **AutoCloseable**, which means that you can use a **try**-with-resources block to manage a socket.

The following program provides a simple **Socket** example. It opens a connection to a **"whois"** port (port 43) on the InterNIC server, sends the command-line argument down the socket, and then prints the data that is returned. InterNIC will try to look up the argument as a registered Internet domain name, and then send back the IP address and contact information for that site.

```
// Demonstrate Sockets.
      import java.net.*;
      import java.io.*;
      class Whois {
      public static void main(String args[]) throws Exception {
          int c;

        // Create a socket connected to internic.net, port 43.
      Socket s = new Socket("whois.internic.net", 43);

        // Obtain input and output streams.
      InputStream in = s.getInputStream();
      OutputStream out = s.getOutputStream();
      // Construct a request string.
      String str = (args.length == 0 ? "MHProfessional.com" : args[0]) + "\n";
      // Convert to bytes.
      byte buf[] = str.getBytes();
         Send request. out.write(buf);
          //Read and display response.
         while ((c = in.read()) != -1) { System.out.print((char) c);
       }

      s.close();
      }
      }
```

If, for example, you obtained information about **MHProfessional.com**, you'd get something similar to the following:

```
Whois Server Version 2.0
Domain names in the .com and .net domains can now be registered
with many different competing registrars. Go to http://www.internic.net for
detailed information.

Domain Name: MHPROFESSIONAL.COM Registrar: CSC CORPORATE DOMAINS,
INC. Whois Server: whois.corporatedomains.com Referral URL:
http://www.cscglobal.com Name Server: NS1.MHEDU.COM
Name Server: NS2.MHEDU.COM
```

Here is how the program works. First, a **Socket** is constructed that specifies the host name "whois.internic.net" and the port number 43. **Internic.net** is the InterNIC web site that handles whois requests. Port 43 is the whois port. Next, both input and output streams are opened on the socket. Then, a string is

constructed that contains the name of the web site you want to obtain information about. In this case, if no web site is specified on the command line, then "MHProfessional.com" is used. The string is converted into a **byte** array and then sent out of the socket. The response is read by inputting from the socket, and the results are displayed. Finally, the socket is closed, which also closes the I/O streams.
In the preceding example, the socket was closed manually by calling **close( )**. If you are using JDK 7 or later, then you can use a **try**-with-resources block to automatically close the socket. For example, here is another way to write the **main( )** method of the previous program:

```
// Use try-with-resources to close a socket.

public static void main(String args[]) throws Exception { int c;

    //Create a socket connected to internic.net, port 43. Manage this

    //socket with a try-with-resources block.

try ( Socket s = new Socket("whois.internic.net", 43) ) {

    //Obtain input and output streams.
    InputStream in = s.getInputStream();
    OutputStream out = s.getOutputStream();
//    Construct a request string.
String str = (args.length == 0 ? "MHProfessional.com" : args[0]) + "\n"; //
//Convert to bytes.
byte buf[] = str.getBytes();
//    Send request.
out.write(buf);
    //Read and display response.
    while ((c = in.read()) != -1) { System.out.print((char) c);
}
}
// The socket is now closed.
}
```

In this version, the socket is automatically closed when the **try** block ends.

So the examples will work with earlier versions of Java and to clearly illustrate when a network resource can be closed, subsequent examples will continue to call **close( )** explicitly. However, in your own code, you should consider using automatic resource management since it offers a more streamlined approach. One other point: In this version, exceptions are still thrown out of **main( )**, but they could be handled by adding **catch** clauses to the end of the **try**-with-resources block

Java DatagramSocket class

Java DatagramSocket class represents a connection-less socket for sending and receiving datagram packets.

A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

Commonly used Constructors of DatagramSocket class

- **DatagramSocket() throws SocketEeption:** it creates a datagram socket and binds it with the available Port Number on the localhost machine.
- **DatagramSocket(int port) throws SocketEeption:** it creates a datagram socket and binds it with the given Port Number.
- **DatagramSocket(int port, InetAddress address) throws SocketEeption:** it creates a datagram socket and binds it with the specified port number and host address.

Java DatagramPacket class

Java DatagramPacket is a message that can be sent or received. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

Commonly used Constructors of DatagramPacket class

- **DatagramPacket(byte[] barr, int length):** it creates a datagram packet. This constructor is used to receive the packets.
- **DatagramPacket(byte[] barr, int length, InetAddress address, int port):** it creates a datagram packet. This constructor is used to send the packets.

Example of Sending DatagramPacket by DatagramSocket

```
//DSender.java
import java.net.*;
public class DSender{
  public static void main(String[] args) throws Exception {
    DatagramSocket ds = new DatagramSocket();
    String str = "Welcome java";
    InetAddress ip = InetAddress.getByName("127.0.0.1");
```

```
    DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(), ip, 30
00);
    ds.send(dp);
    ds.close();
  }
}
```

Example of Receiving DatagramPacket by DatagramSocket

```
//DReceiver.java
import java.net.*;
public class DReceiver{
  public static void main(String[] args) throws Exception {
    DatagramSocket ds = new DatagramSocket(3000);
    byte[] buf = new byte[1024];
    DatagramPacket dp = new DatagramPacket(buf, 1024);
    ds.receive(dp);
    String str = new String(dp.getData(), 0, dp.getLength());
    System.out.println(str);
    ds.close();
  }
}
```

1. Which of these package contains classes and interfaces for networking?
a) java.io
b) java.util
c) java.net
d) java.network
Answer: c
2. Which of these is a protocol for breaking and sending packets to an address across a network?
a) TCP/IP
b) DNS
c) Socket
d) Proxy Server
Answer: a
Explanation: TCP/IP – Transfer control protocol/Internet Protocol is used to break data into small packets an send them to an address across a network.
3. How many ports of TCP/IP are reserved for specific protocols?
a) 10
b) 1024
c) 2048
d) 512
Answer: b
4. How many bits are in a single IP address?
a) 8
b) 16
c) 32
d) 64
Answer: c
5. Which of these is a full form of DNS?
a) Data Network Service
b) Data Name Service
c) Domain Network Service
d) Domain Name Service
Answer: d
6. Which of these class is used to encapsulate IP address and DNS?
a) DatagramPacket
b) URL
c) InetAddress
d) ContentHandler
Answer: c
Explanation: InetAddress class encapsulate both IP address and DNS, we can interact with this class by using name of an IP host.
7. What is the output of this program?

```
import java.net.*;
class networking
{
   public static void main(String[] args) throws UnknownHostException
   {
      InetAddress obj1 = InetAddress.getByName("sanfoundry.com");
      InetAddress obj2 = InetAddress.getByName("sanfoundry.com");
```

```
        boolean x = obj1.equals(obj2);
        System.out.print(x);
    }
}
```

a) 0
b) 1
c) true
d) false
Answer: c
Output:

```
$ javac networking.java
$ java networking
true
```

8. What is the output of this program?

```
import java.net.*;
public class networking
{
   public static void main(String[] args) throws UnknownHostException
   {
      InetAddress obj1 = InetAddress.getByName("cisco.com");
      InetAddress obj2 = InetAddress.getByName("sanfoundry.com");
      boolean x = obj1.equals(obj2);
      System.out.print(x);
   }
}
```

a) 0
b) 1
c) true
d) false
Answer: d
Explanation: InetAddress obj1 = InetAddress.getByName(“cisco.com”); creates object obj1 having DNS and IP address of cisco.com, InetAddress obj2 = InetAddress.getByName(“sanfoundry.com”); creates obj2 having DNS and IP address of sanfoundry.com, since both these address point to two different locations false is returned by obj1.equals(obj2);.
Output:

```
$ javac networking.java
$ java networking
False
```

9. What is the output of this program?

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

```
    {
        try
        {
            URL url=new URL("https://www.sanfoundry.com/java-mcq");
            System.out.println("Protocol: "+url.getProtocol());
            System.out.println("Host Name: "+url.getHost());
            System.out.println("Port Number: "+url.getPort());
        } catch(Exception e){System.out.println(e);}
    }
}
```

a) Protocol: http
b) Host Name: www.sanfoundry.com
c) Port Number: -1
d) all of the mentioned
Answer: d
Explanation: getProtocol() give protocol which is http
getUrl() give name domain name
getPort() Since we have not explicitly set the port, default value that is -1 is printed.

10. What is the output of this program?

```
import java.net.*;
class networking
{
    public static void main(String[] args) throws UnknownHostException
    {
        InetAddress obj1 = InetAddress.getByName("cisco.com");
        System.out.print(obj1.getHostName());
    }
}
```

a) cisco
b) cisco.com
c) www.cisco.com
d) none of the mentioned
Answer: b
Explanation: None.
Output:
$ javac networking.java
$ java networking
cisco.com

11. **Which class is used to create servers that listen for either local client or remote client programs?**

a. ServerSockets
b. httpServer
c. httpResponse
d. None of the above
ANSWER: ServerSockets

12. Which constructor of DatagramSocket class is used to creates a datagram socket and binds it with the given Port Number?

a. DatagramSocket(int port)
b. DatagramSocket(int port, InetAddress address)

c. DatagramSocket()
d. None of the above
ANSWER: DatagramSocket(int port, InetAddress address)
13. Which methods are commonly used in ServerSocket class?
a. public OutputStream getOutputStream()
b. public Socket accept()
c. public synchronized void close()
d. None of the above
ANSWER: public Socket accept()
14. Which classes are used for connection-less socket programming?
a. DatagramSocket
b. DatagramPacket
c. Both A & B
d. None of the above
ANSWER: Both A & B
15. Which method of URL class represents a URL and it has complete set of methods to manipulate URL in Java?
a. java.net.URL
b. java.net.URLConnection
c. Both A & B
d. None of the above
ANSWER: java.net.URL
16. The DatagramSocket and DatagramPacket classes are not used for connection-less socket programming.
a. True
b. False
ANSWER: False