UNIT-11: MULTICAST

- IP multicast in Java allows you to send and receive multicast packets over an IP network. Multicast is a communication method where **a single packet can be sent to multiple recipients simultaneously**.
- IP multicast is a communication method that allows the transmission of data from **one sender to multiple recipients over an IP network**. It is designed to **efficiently distribute data to a group of interested receivers, reducing network traffic and improving scalability**.

Importance:

- **Efficient Data Distribution**: IP multicasting allows the transmission of data to multiple recipients simultaneously, reducing network traffic and conserving bandwidth compared to unicast transmissions.
- **Scalability**: IP multicasting enables applications to scale efficiently to a large number of receivers without overloading the network or the sender's resources.
- **Real-Time Applications**: IP multicasting is essential for real-time applications, such as multimedia streaming, online gaming, teleconferencing, and financial data dissemination, where timely and synchronized data delivery to multiple recipients is crucial.
- **Content Delivery Networks (CDNs)**: CDNs utilize IP multicasting to efficiently distribute popular content to multiple edge servers, reducing bandwidth usage and relieving the load on origin servers.
- **Network Efficiency**: IP multicasting optimizes network utilization by eliminating the need for point-to-point connections between the sender and each receiver, resulting in reduced network congestion, lower latency, and improved overall network performance.

UNIT-11: MULTICAST

Importance

- **Internet of Things (IoT):** IP multicasting plays a significant role in IoT applications, where efficient data dissemination to a large number of devices is required, allowing updates, notifications, and control messages to be efficiently distributed.

- **Multicast-Based Routing Protocols**: IP multicasting relies on multicast-specific routing protocols, such as Protocol Independent Multicast (PIM), which dynamically establish and maintain multicast group membership information in routers, enabling efficient forwarding of multicast packets in the network.

- IP multicasting is important for efficient data distribution, scalability, real-time applications, content delivery networks, network efficiency, IoT applications, and relies on specialized multicast-based routing protocols. It provides significant benefits in terms of **reduced network traffic, improved scalability, and enhanced performance for a variety of applications and network environments**.

- Please note that for successful communication, the sender and receiver programs must use the **same multicast group address and port**. Additionally, the multicast packets are limited to **the same network segment or subnet**.

UNIT-11: MULTICAST

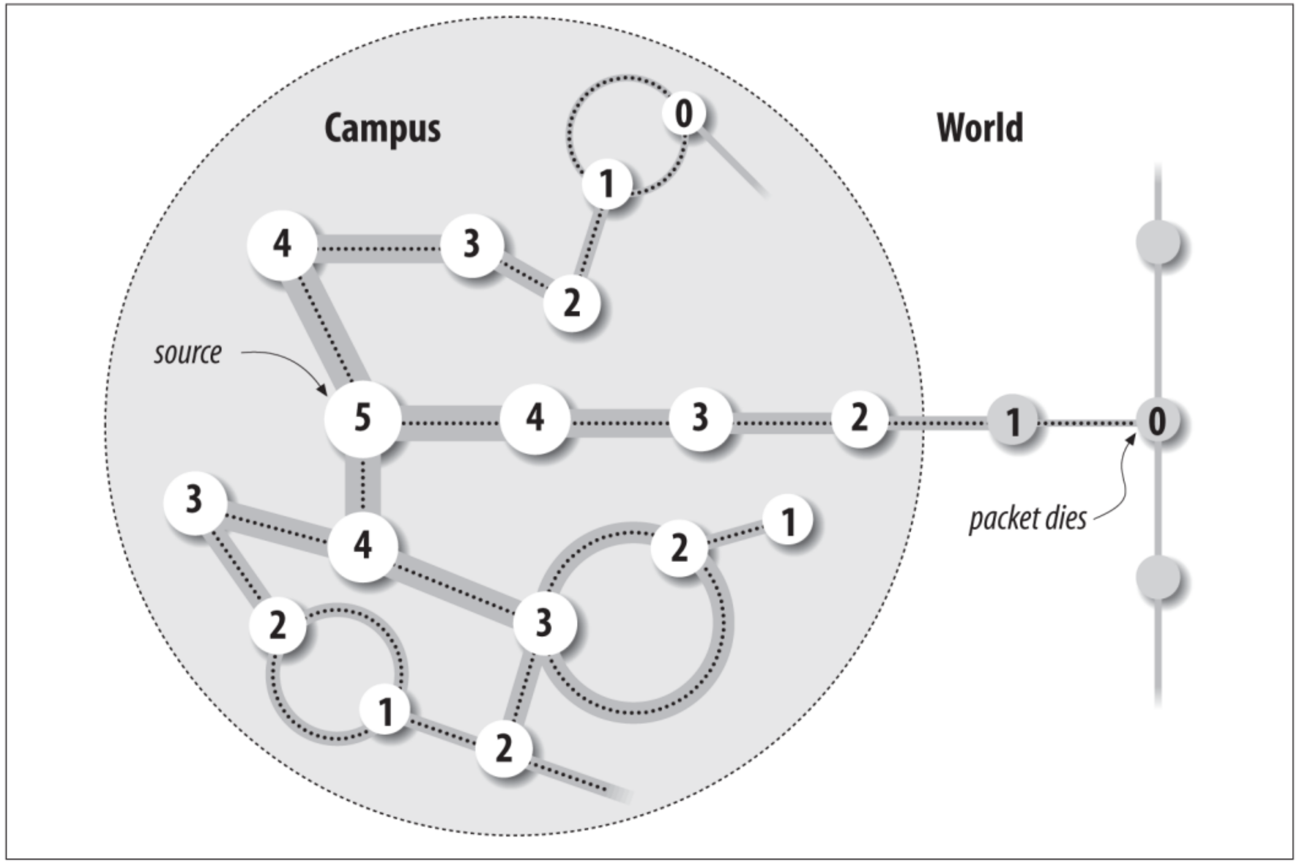
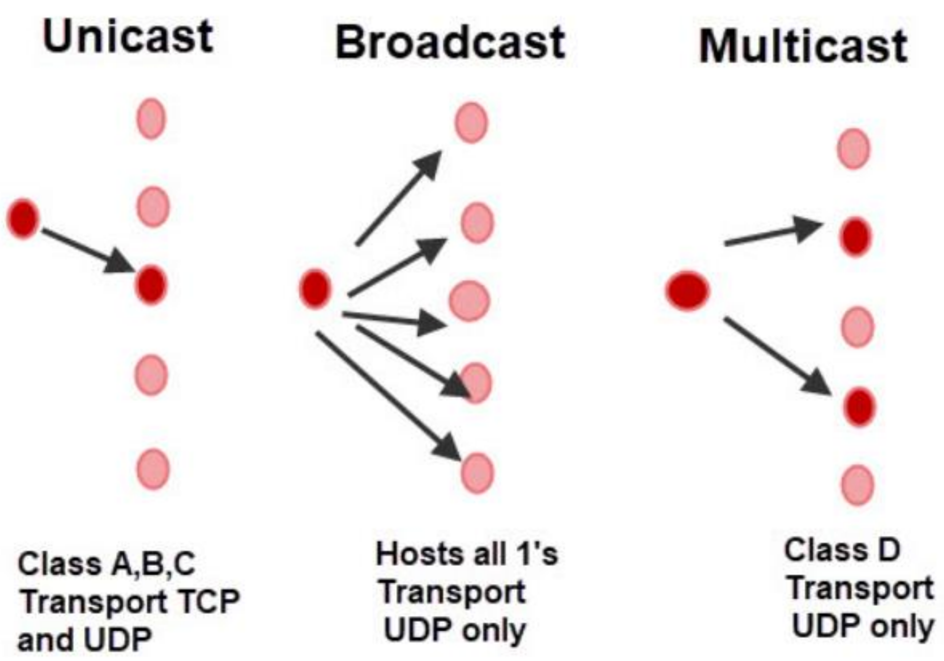


Figure 13-2. Coverage of a packet with a TTL of five



Class A,B,C
Transport TCP
and UDP

Hosts all 1's
Transport
UDP only

Class D
Transport
UDP only

UNIT-11: MULTICAST

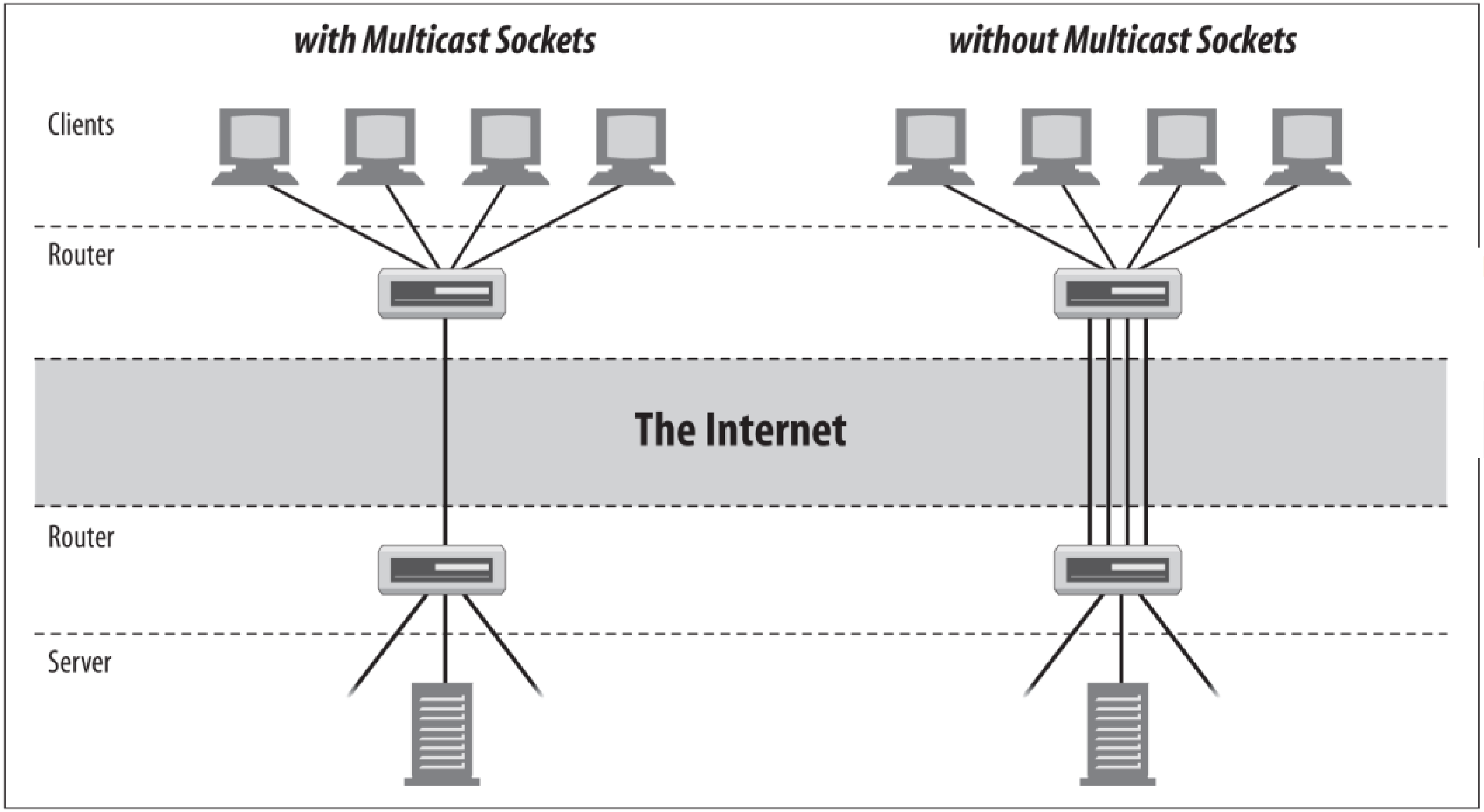


Figure 13-3. With and without multicast sockets

Multicast Addresses

IPv4 Multicast addresses use the reserved **class D** address range:

```
224.0.0.0 through 239.255.255.255
```

UNIT-11: MULTICAST

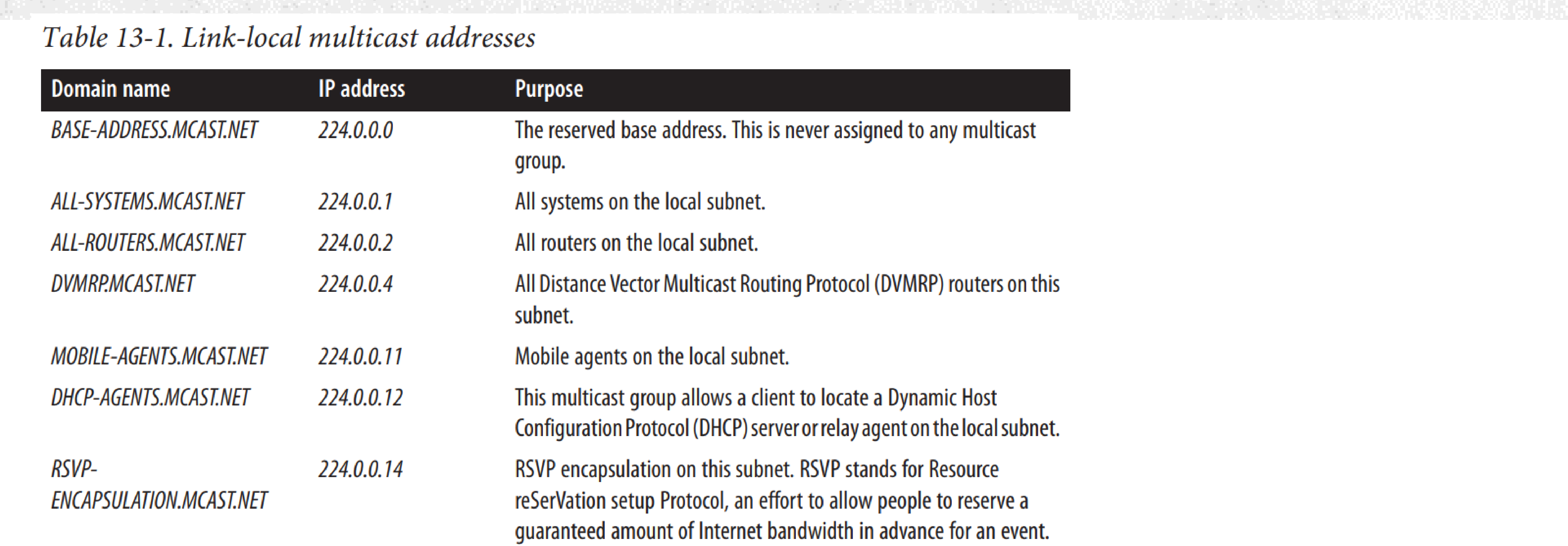


Table 13-1. Link-local multicast addresses

Domain name	IP address	Purpose
BASE-ADDRESS.MCAST.NET	*224.0.0.0*	The reserved base address. This is never assigned to any multicast group.
ALL-SYSTEMS.MCAST.NET	*224.0.0.1*	All systems on the local subnet.
ALL-ROUTERS.MCAST.NET	*224.0.0.2*	All routers on the local subnet.
DVMRP.MCAST.NET	*224.0.0.4*	All Distance Vector Multicast Routing Protocol (DVMRP) routers on this subnet.
MOBILE-AGENTS.MCAST.NET	*224.0.0.11*	Mobile agents on the local subnet.
DHCP-AGENTS.MCAST.NET	*224.0.0.12*	This multicast group allows a client to locate a Dynamic Host Configuration Protocol (DHCP) server or relay agent on the local subnet.
RSVP-ENCAPSULATION.MCAST.NET	*224.0.0.14*	RSVP encapsulation on this subnet. RSVP stands for Resource reSerVation setup Protocol, an effort to allow people to reserve a guaranteed amount of Internet bandwidth in advance for an event.

MULTICASTSOCKET CLASS: CONSTRUCTOR

- The **MulticastSocket** class in Java provides functionality for sending and receiving multicast packets. It extends the DatagramSocket class and adds **specific methods for multicast communication**. Here's an overview of the construction and some commonly used methods of the MulticastSocket class:

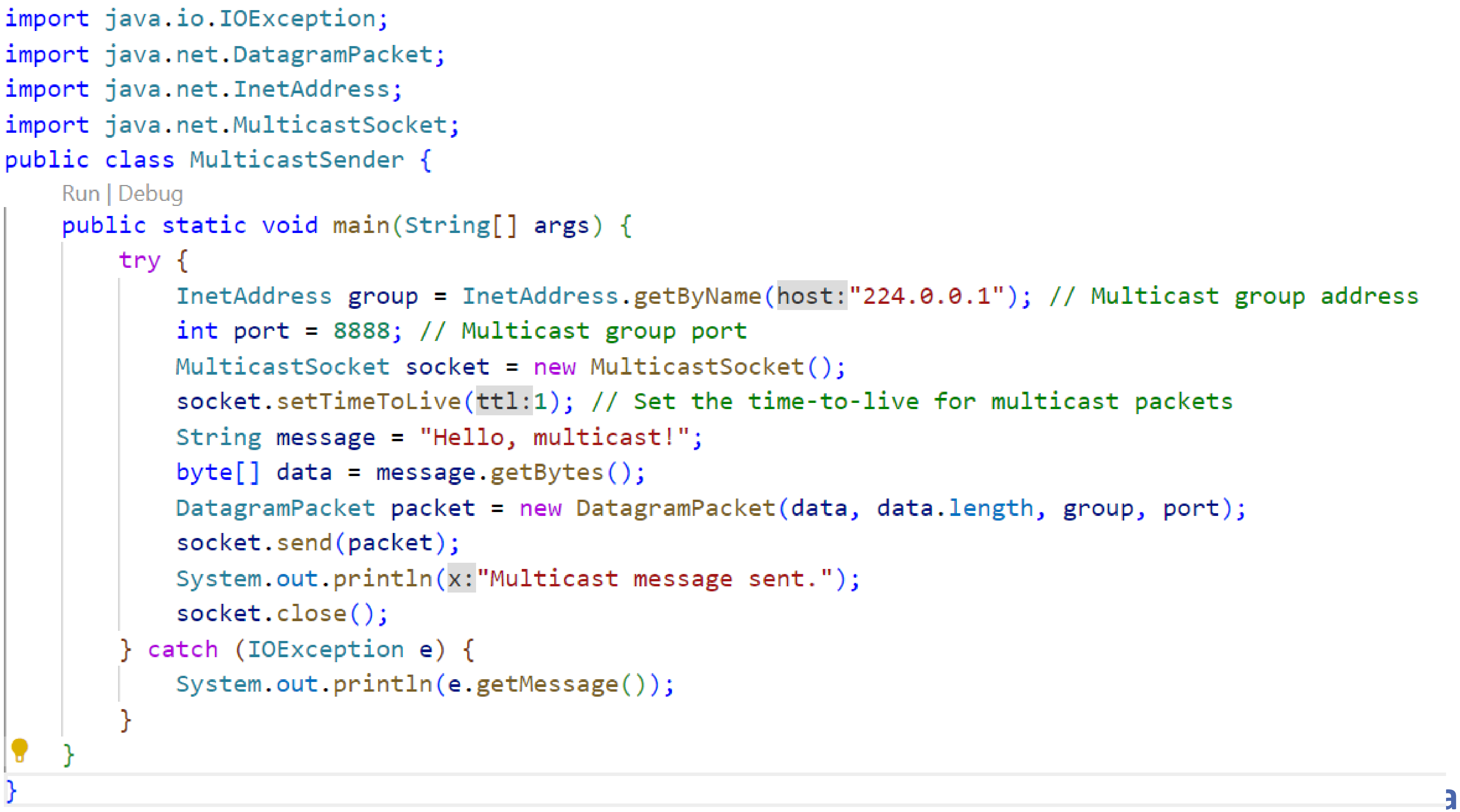
Construction:

1. MulticastSocket(): Constructs a new MulticastSocket that is bound to **any available port on the local host machine**.
2. MulticastSocket(int port): Constructs a new MulticastSocket bound to the **specified port on the local host machine**.

Commonly Used Methods:

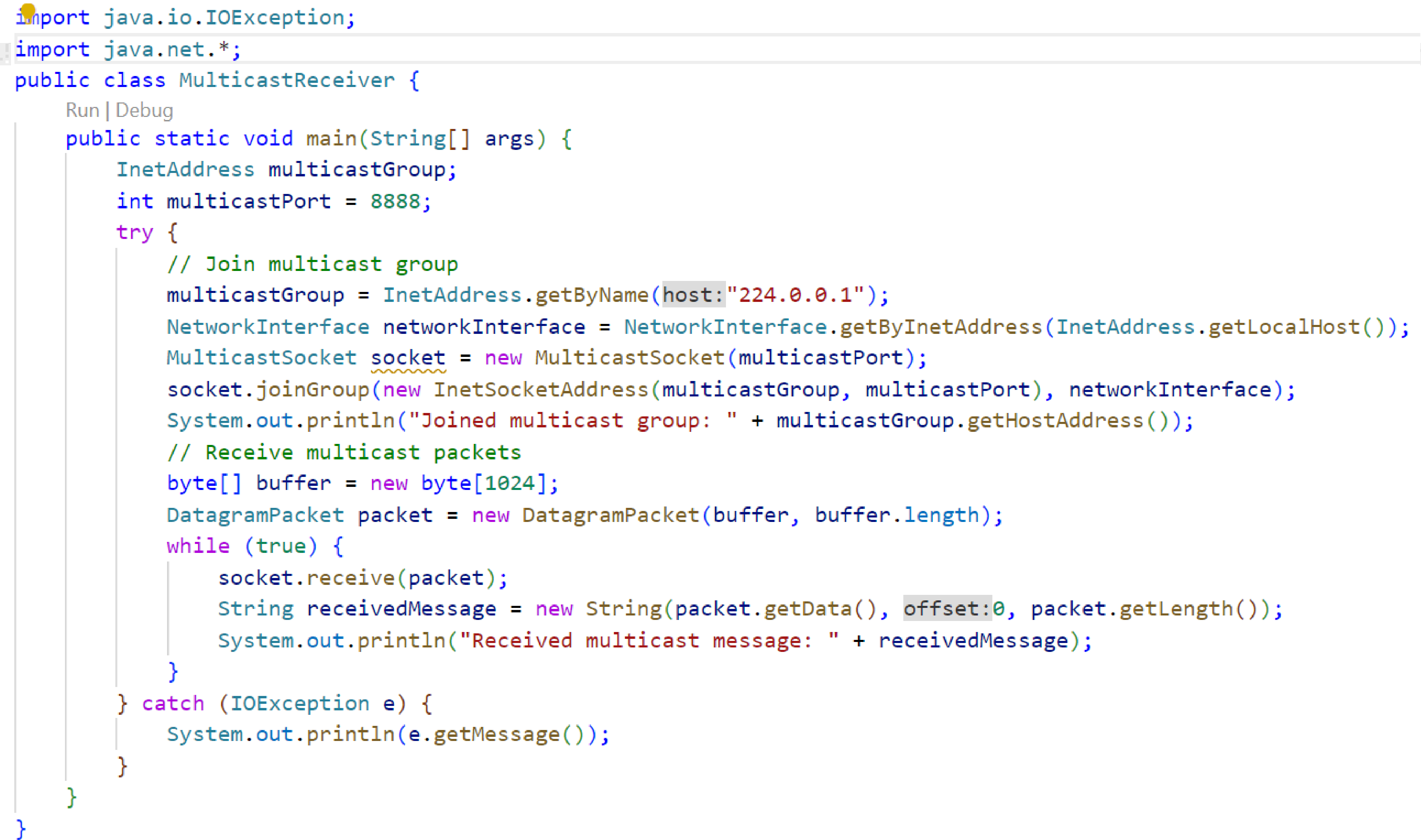
1. joinGroup(InetAddress multicastAddress): Joins the specified multicast group by joining the multicast address. The socket will start receiving multicast packets sent to this group.
2. leaveGroup(InetAddress multicastAddress): Leaves the specified multicast group by leaving the multicast address. The socket will stop receiving multicast packets sent to this group.
3. setInterface(InetAddress address): Sets the network interface to use for multicast communication. The multicast packets will be sent or received through this interface.
4. setTimeToLive(int ttl): Sets the time-to-live (TTL) value for multicast packets sent from this socket. The TTL determines how far the multicast packets can propagate in the network.
5. setLoopbackMode(boolean enable): Sets the loopback mode for multicast packets. If enabled, the socket will receive the multicast packets sent by itself.
6. receive(DatagramPacket packet): Receives a datagram packet from this socket. The received packet's data, length, and source address will be populated into the provided DatagramPacket object.
7. send(DatagramPacket packet, byte ttl): Sends a datagram packet from this socket to the specified destination address with the specified time-to-live (TTL) value.
8. close(): Closes the socket and frees any resources associated with it.

UNIT-11: MULTICASTING SENDER



```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;
public class MulticastSender {
    Run | Debug
    public static void main(String[] args) {
        try {
            InetAddress group = InetAddress.getByName(host:"224.0.0.1"); // Multicast group address
            int port = 8888; // Multicast group port
            MulticastSocket socket = new MulticastSocket();
            socket.setTimeToLive(ttl:1); // Set the time-to-live for multicast packets
            String message = "Hello, multicast!";
            byte[] data = message.getBytes();
            DatagramPacket packet = new DatagramPacket(data, data.length, group, port);
            socket.send(packet);
            System.out.println(x:"Multicast message sent.");
            socket.close();
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

UNIT-11: MULTICASTING RECOVER



```
import java.io.IOException;
import java.net.*;
public class MulticastReceiver {
    Run | Debug
    public static void main(String[] args) {
        InetAddress multicastGroup;
        int multicastPort = 8888;
        try {
            // Join multicast group
            multicastGroup = InetAddress.getByName(host:"224.0.0.1");
            NetworkInterface networkInterface = NetworkInterface.getByInetAddress(InetAddress.getLocalHost());
            MulticastSocket socket = new MulticastSocket(multicastPort);
            socket.joinGroup(new InetSocketAddress(multicastGroup, multicastPort), networkInterface);
            System.out.println("Joined multicast group: " + multicastGroup.getHostAddress());
            // Receive multicast packets
            byte[] buffer = new byte[1024];
            DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
            while (true) {
                socket.receive(packet);
                String receivedMessage = new String(packet.getData(), offset:0, packet.getLength());
                System.out.println("Received multicast message: " + receivedMessage);
            }
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```