Java provides a robust and versatile networking framework that allows you to create networked applications. Java networking is commonly used to develop client-server applications, where one program (the client) communicates with another program (the server) over a network. Java's networking capabilities are based on classes and interfaces in the `java.net` package. Below are two examples to illustrate Java networking.

Example 1: Simple Client-Server Communication

In this example, we'll create a simple Java client-server application where the client sends a message to the server, and the server responds.

Server Side (Server.java):
```java
import java.io.*;
import java.net.*;

public class Server {
    public static void main(String[] args) throws IOException {
        // Create a ServerSocket that listens on port 12345
        ServerSocket serverSocket = new ServerSocket(12345);
        System.out.println("Server is waiting for a client to connect...");

        // Accept client connection
        Socket clientSocket = serverSocket.accept();
        System.out.println("Client connected!");

        // Create input and output streams for communication
        BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

        // Read data from the client
        String message = in.readLine();
        System.out.println("Client says: " + message);

        // Send a response to the client
        out.println("Hello, Client!");

        // Close the connections
        in.close();
        out.close();
        clientSocket.close();
        serverSocket.close();
    }
```

```
}
```

Client Side (Client.java):
```java
import java.io.*;
import java.net.*;

public class Client {
    public static void main(String[] args) throws IOException {
        // Create a socket to connect to the server on port 12345
        Socket clientSocket = new Socket("localhost", 12345);

        // Create input and output streams for communication
        BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

        // Send a message to the server
        out.println("Hello, Server!");

        // Read the server's response
        String response = in.readLine();
        System.out.println("Server says: " + response);

        // Close the connections
        in.close();
        out.close();
        clientSocket.close();
    }
}
```

Compile and run the server and client in separate terminals. The client sends a message to the server, and the server responds.

Example 2: URL Connections

Java also provides a way to interact with web resources using URL connections. Here's an example of how to fetch the content of a web page.

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

```
public class URLReader {
    public static void main(String[] args) throws IOException {
        URL url = new URL("https://www.example.com");
        BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()));

        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            System.out.println(inputLine);
        }
        in.close();
    }
}
```

In this example, we create a URL object representing a web page, open a connection to it, and read its content.

These examples provide a basic introduction to Java networking, but Java's networking capabilities are quite extensive and can be used for various network-related tasks.