Java provides a powerful and flexible networking framework that allows you to create networked applications easily. Here are some advantages of Java networking along with code samples to illustrate them:

1. **Platform Independence**: Java networking is platform-independent, meaning that you can write networked applications once and run them on any platform that supports Java without modification.

2. **Socket Programming**: Java provides the `java.net` package for socket programming, making it relatively straightforward to create client-server applications. Here's a simple example:

   Server Example:
```java
   import java.io.*;
   import java.net.*;

   public class Server {
       public static void main(String[] args) {
           try (ServerSocket serverSocket = new ServerSocket(8080)) {
               System.out.println("Server is listening on port 8080");
               Socket clientSocket = serverSocket.accept();
               PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
               out.println("Hello, client!");
           } catch (IOException e) {
               e.printStackTrace();
           }
       }
   }
```

   Client Example:
```java
   import java.io.*;
   import java.net.*;

   public class Client {
       public static void main(String[] args) {
           try (Socket socket = new Socket("localhost", 8080);
                BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()))) {
               String response = in.readLine();
               System.out.println("Server says: " + response);
           } catch (IOException e) {
               e.printStackTrace();
```

```
        }
    }
}
```

3. **High-Level Abstractions**: Java provides high-level abstractions for network communication. For example, the `URL` and `URLConnection` classes make it easy to work with web resources. Here's an example of reading content from a URL:

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

public class URLReader {
    public static void main(String[] args) {
        try {
            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();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

4. **Security**: Java networking is designed with security in mind. It supports SSL/TLS for secure communication and provides security features like authentication and access control.

5. **Concurrency**: Java provides support for multi-threading and concurrency, making it easy to create networked applications that can handle multiple clients simultaneously.

6. **Community Support**: Java has a large and active community, which means you can find a wealth of resources, libraries, and frameworks for various networking tasks.

7. **Scalability**: Java's networking capabilities make it suitable for both small and large-scale applications, from simple client-server interactions to enterprise-level distributed systems.

8. **Cross-Platform Compatibility**: Java networking is not limited to traditional client-server models; you can create peer-to-peer, multicast, and distributed systems that work across different operating systems.

9. **Robustness**: Java's exception handling and error reporting mechanisms make it easier to write robust networked applications that can gracefully handle network failures and recover from errors.

Overall, Java's networking capabilities are robust and versatile, making it a popular choice for building networked applications, whether you're developing simple client-server interactions or complex distributed systems.