

A decorative background consisting of a dark blue vertical bar on the left side, with a small orange square positioned on the boundary between the blue bar and the main white area. The rest of the background is white.

TUTORIAL

JAVA AS - HTTP CLIENT COMPOSITION ENVIROMENT 7.1

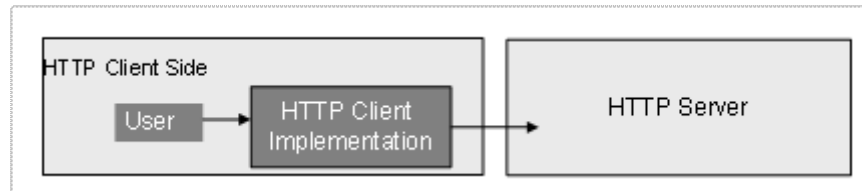
FABIO HAIDER
2008

CONTEÚDO :

1 HTTP Client.....	3
1.1 Arquitetura.....	3
1.2 Usando HTTP Client.....	3
1.2.1 Usuário.....	3
1.2.2 Implementação HTTP Client.....	3
1.3 Recursos.....	4
1.4 Exemplo.....	5
1.4.1 Especificando Host e Port e Recuperando uma Simples Requisição	5

1 HTTP Client

1.1 Arquitetura



HTTP client implementa partes dos protocolos HTTP 1.0 e HTTP 1.1, incluindo:

- Métodos de requisição HTTP HEAD, GET, POST, CONNECT, DELETE, OPTIONS, TRACE e PUT.
- Métodos de extensão WEBDAV COPY, LOCK, MKCOL, MOVE, PROPFIND, PROPPATCH e UNLOCK.
- Verificação automática de autorizações, redirecionamento de requisições, cookies, pool de conexões. Tudo isto é feito pelo Gerenciador de Conexão.

1.2 Usando HTTP Client

1.2.1 Usuário

Para implementar HTTP client, é necessário:

- Instanciar a classe HTTPClient
- Configurar as propriedades para a instância HTTP Client
- Conectar a implementação HTTP client e executar os métodos desejados
- Processar os resultados vindos da requisição

1.2.2 Implementação HTTP Client

A implementação HTTP client contém métodos processadores HTTP, no qual checa a autenticação ou redirecionamento que podem ser executados, então a conexão ao Gerenciador de Conexão é estabelecida. O Gerenciador de Conexão HTTP escolhe entre abrir uma nova conexão ou pegar um conexão do pool de conexões e conecta ao servidor HTTP. Tendo recebido o recurso, o processador de método HTTP envia a resposta ao usuário.

1.3 Recursos

- Gerenciamento de Cookies – permite ao servidor HTTP definir um cookie ou recuperar um existente. O método tem o estado e o cookie detém o estado. Use a API para recuperar este estado.
- Pool de Conexões – Permite que múltiplos usuários compartilhem um conjunto de objetos de conexões provendo acesso para algum recurso de banco de dados. Isto é automaticamente executado pelo Gerenciador de Conexão.
- Métodos Customizados – podem ser criados e enviados métodos HTTP customizados como uma requisição ao servidor HTTP.
- Gerenciadores de Conexão Plugáveis – pode ser adicionado algoritmos para Gerenciadores de Conexão customizados e trocar o Gerenciador de Conexão padrão pela nova implementação.
- Suporte à Conexões Keep Alive – permite conexões persistentes. Por padrão a conexão está sempre aberta.
- Suporte ao timeout da Conexão – pode ser definido o tempo limite para poder recuperar uma conexão no Gerenciador de Conexão.
- Suporte ao timeout da Requisição – pode ser definido a propriedade socket-timeout para lançar uma exceção por tempo limite, não ocorrendo a resposta.
- Suporte ao timeout para Conexões idle – usando a API pode ser definido o tempo limite na qual todas as conexões idle são automaticamente fechadas.
- Logging Integrado – usando a Logging API pode ser gerado automaticamente logs de diagnóstico e informação de segurança.
- Suporte ao SSL - suporte ao protocolo SSL, permitindo aos navegadores Web e servidores, comunicarem-se em uma conexão segura.
- Autenticação Basic – suporte para autenticação Basic, onde é requerida uma instância do objeto UserPassCredentials que após ser preenchido com o nome de usuário e senha, é enviado ao cliente.
- Autenticação Digest – suporte para autenticação Digest, com um pouco mais de segurança que a autenticação Basic. O suporte não transfere a senha livremente através da rede, mas de forma encriptada. É requerida uma instância do objeto UserPassCredentials que após ser preenchido com o nome de usuário e senha, é enviado ao cliente.
- Autenticação NTLM – suporte ao protocolo proprietário que usa nomes do domínio como realm. Clientes podem provar a sua identidade sem enviar a senha para o servidor HTTP.
- Autenticação Proxy – suporte para autenticação Proxy, na qual é semelhante com a autenticação de servidor, com a diferença que as credenciais de cada proxy são armazenadas independentemente.
- WEBDAV – suporte ao Web-Based Distributed Authoring and Versioning, fornecendo funções para os usuários colaborarem, editarem e manusearem arquivos em servidores web remotos.

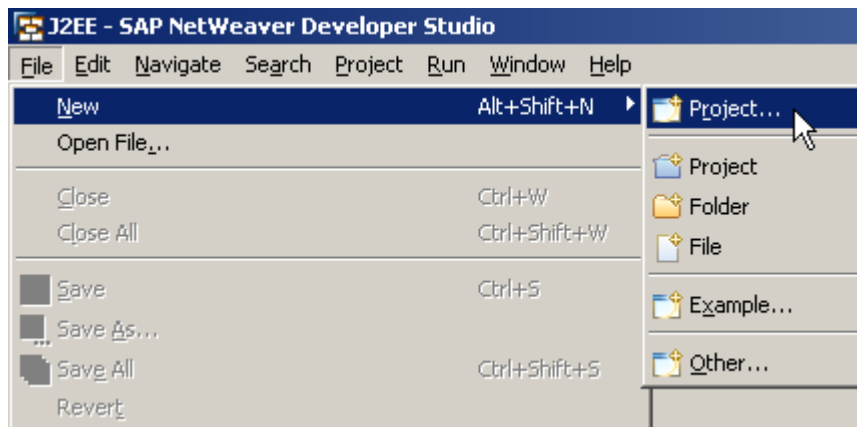
1.4 Exemplo

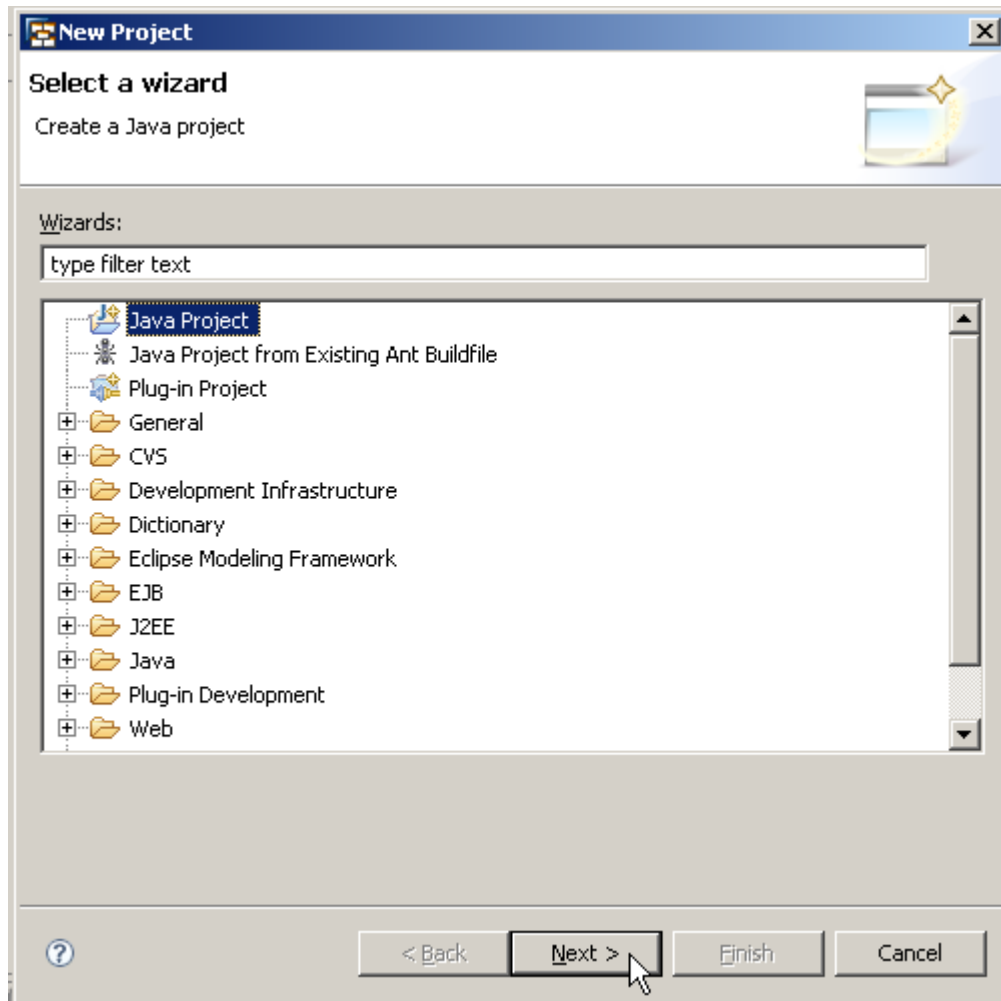
Para rodar os nosso exeplo, será necessário baixar a API Jakarta Commons HTTP Client do seguinte endereço:

<http://linorg.usp.br/apache/httpcomponents/httpcore/binary/httpcomponents-core-4.0-beta1-bin.zip>

Utilizaremos a lib httpcore-4.0-beta1.jar em nosso projeto do NWDS.

1.4.1 Especificando Host e Port e Recuperando uma Simples Requisição





New Java Project
X

Create a Java project

Create a Java project in the workspace or in an external location.

Project name:

Contents

Create new project in workspace

Create project from existing source

Directory:

JRE

Use default JRE (Currently 'jdk1.5.0_13') [Configure JREs...](#)

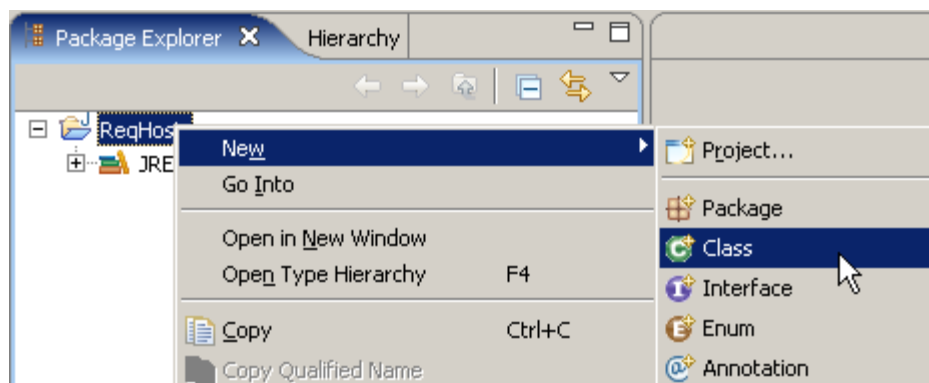
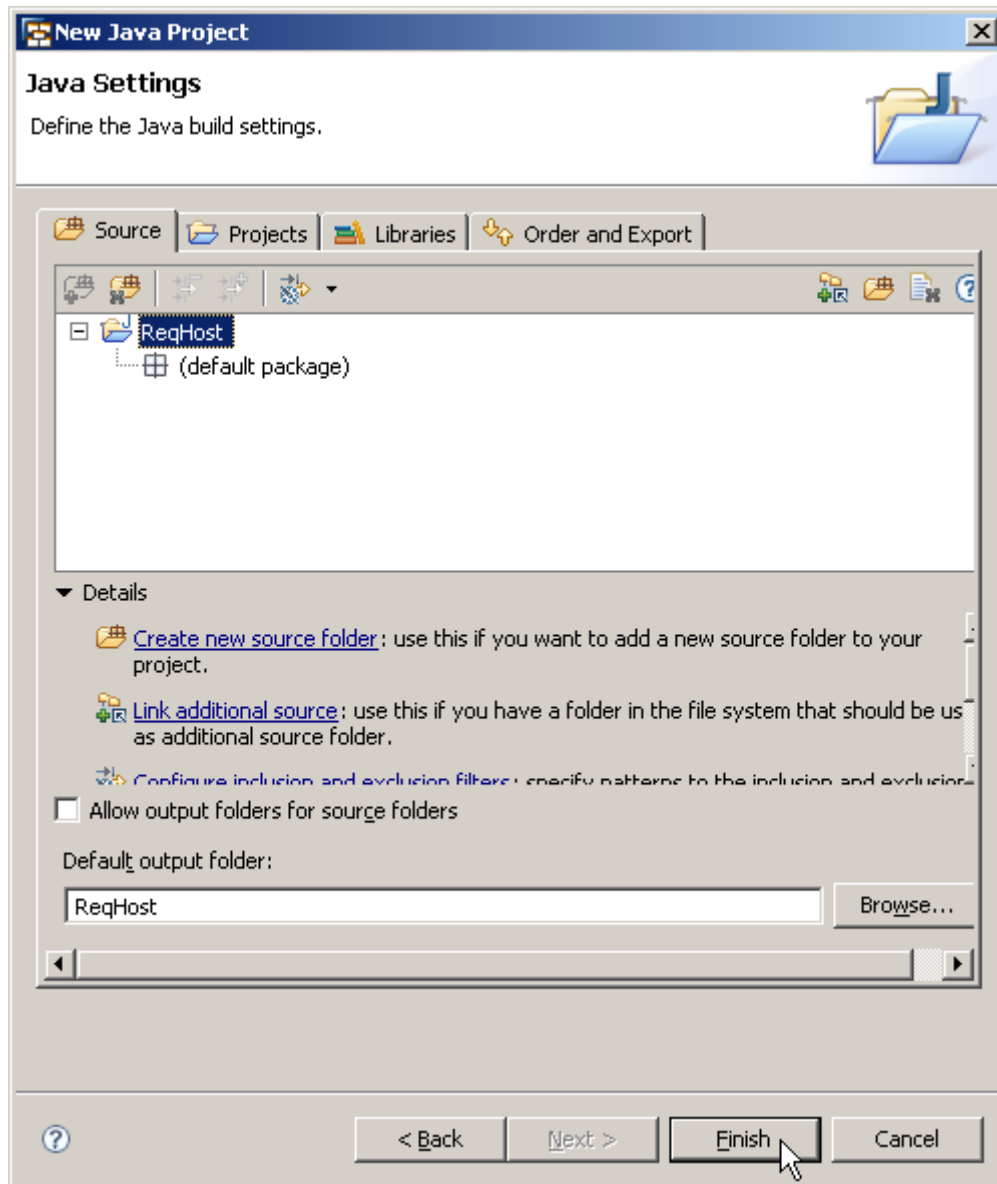
Use a project specific JRE:

Project layout

Use project folder as root for sources and class files

Create separate source and output folders [Configure default...](#)

?



New Java Class
✕

Java Class

⚠ Type name is discouraged. By convention, Java type names usually start with an uppercase letter

Source folder:

Browse...

Package:

Browse...

Enclosing type:

Browse...

Name:

Modifiers:
 public
 default
 private
 protected
 abstract
 final
 static

Superclass:

Browse...

Interfaces:

Which method stubs would you like to create?

- public static void main(String[] args)
- Constructors from superclass
- Inherited abstract methods

Do you want to add comments as configured in the [properties](#) of the current project?

- Generate comments

?

New Java Class
✕

Java Class

Create a new Java class.

Source folder: Browse...

Package: Browse...

Enclosing type: Browse...

Name:

Modifiers: public default private protected
 abstract final static

Superclass: Browse...

Interfaces: Add...
Remove

Which method stubs would you like to create?

public static void main(String[] args)

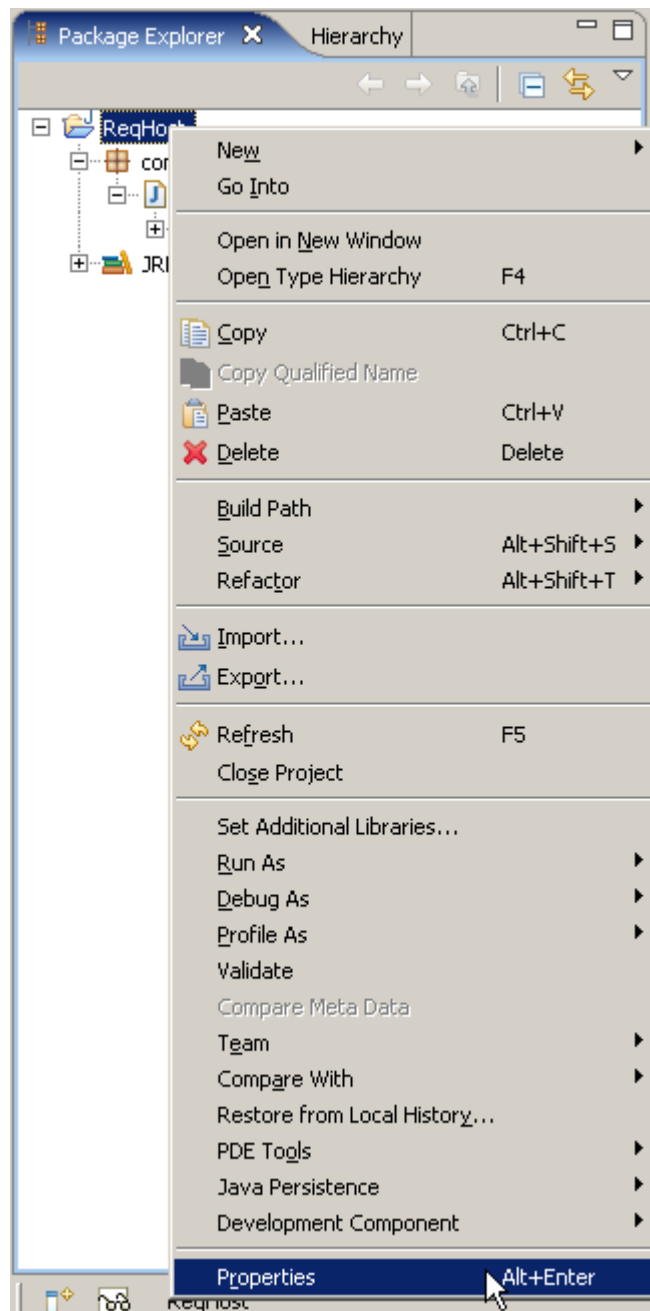
Constructors from superclass

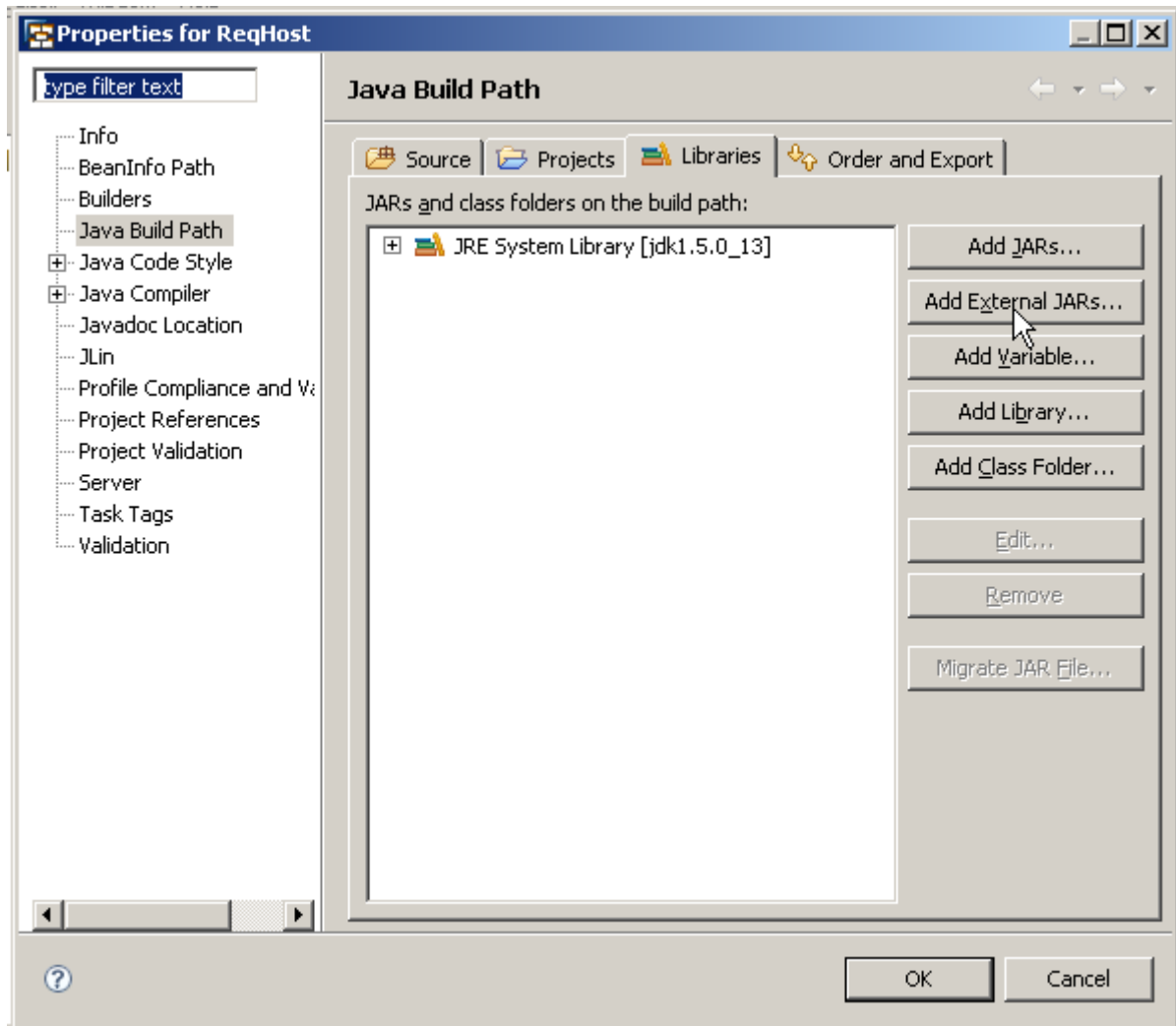
Inherited abstract methods

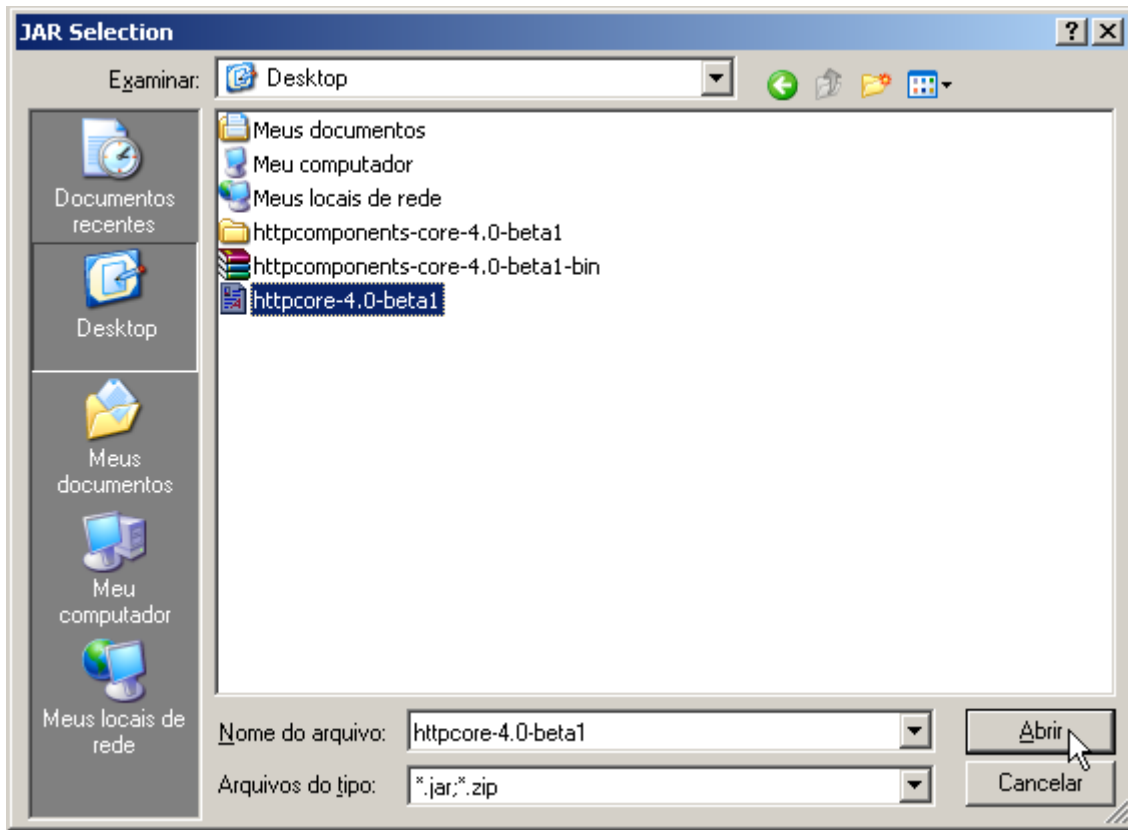
Do you want to add comments as configured in the [properties](#) of the current project?

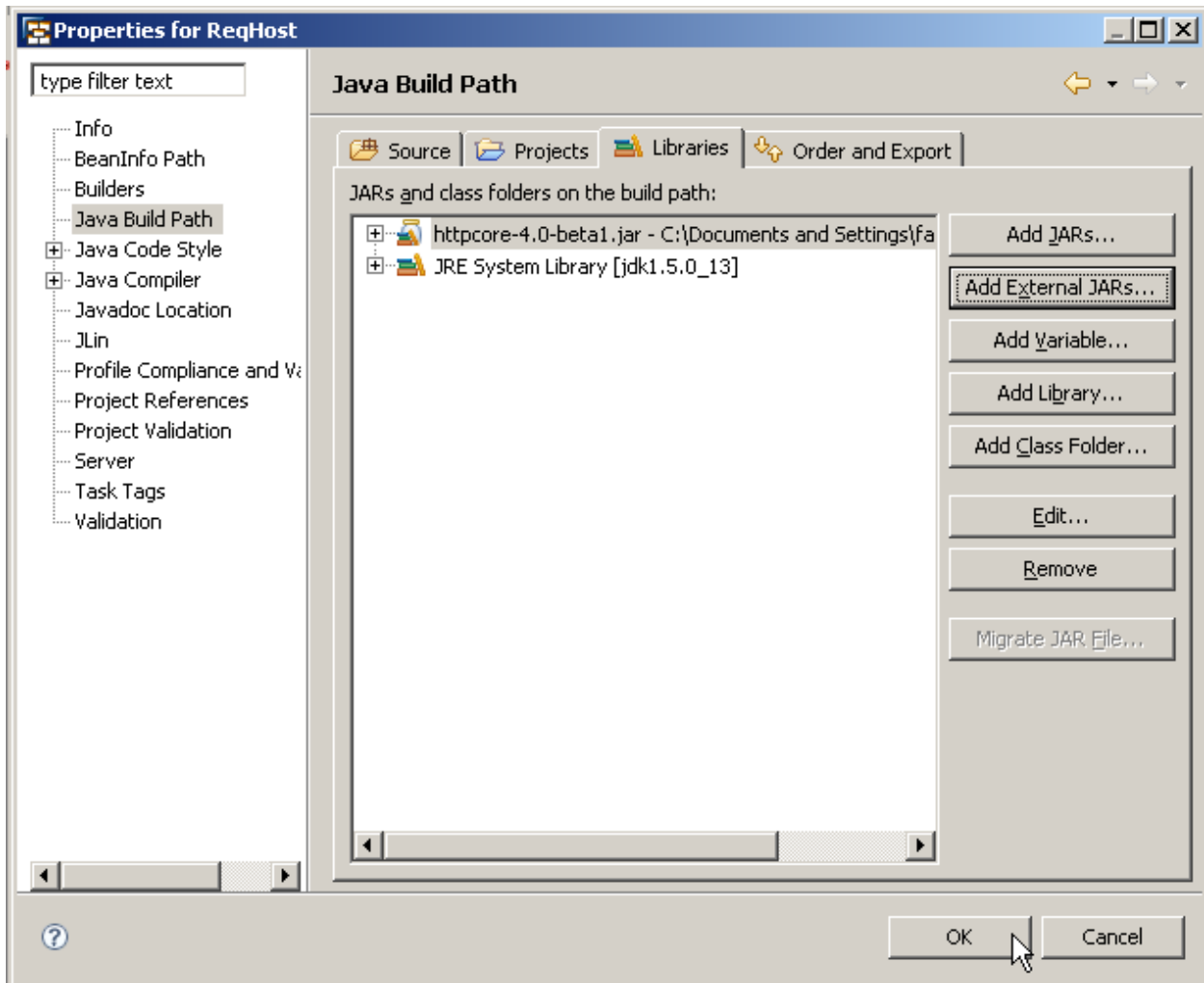
Generate comments

Finish
Cancel









Coloque o seguinte código na classe:

```

package com.tutorial.http.client;

import java.net.Socket;

import org.apache.http.ConnectionReuseStrategy;
import org.apache.http.HttpHost;
import org.apache.http.HttpResponse;
import org.apache.http.HttpVersion;
import org.apache.http.impl.DefaultConnectionReuseStrategy;
import org.apache.http.impl.DefaultHttpClientConnection;
import org.apache.http.params.BasicHttpParams;
import org.apache.http.message.BasicHttpRequest;
import org.apache.http.params.HttpParams;
import org.apache.http.params.HttpProtocolParams;
import org.apache.http.protocol.BasicHttpProcessor;
import org.apache.http.protocol.HttpContext;
import org.apache.http.protocol.BasicHttpContext;
import org.apache.http.protocol.ExecutionContext;

```

```

import org.apache.http.protocol.HttpRequestExecutor;
import org.apache.http.protocol.RequestConnControl;
import org.apache.http.protocol.RequestContent;
import org.apache.http.protocol.RequestExpectContinue;
import org.apache.http.protocol.RequestTargetHost;
import org.apache.http.protocol.RequestUserAgent;
import org.apache.http.util.EntityUtils;

public class RequestHost {

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

        HttpParams params = new BasicHttpParams();
        HttpProtocolParams.setVersion(params, HttpVersion.HTTP_1_1);
        HttpProtocolParams.setContentCharset(params, "UTF-8");
        HttpProtocolParams.setUserAgent(params, "HttpComponents/1.1");
        HttpProtocolParams.setUseExpectContinue(params, true);

        BasicHttpProcessor httpproc = new BasicHttpProcessor();
        // Required protocol interceptors
        httpproc.addInterceptor(new RequestContent());
        httpproc.addInterceptor(new RequestTargetHost());
        // Recommended protocol interceptors
        httpproc.addInterceptor(new RequestConnControl());
        httpproc.addInterceptor(new RequestUserAgent());
        httpproc.addInterceptor(new RequestExpectContinue());

        HttpRequestExecutor httpexecutor = new HttpRequestExecutor();

        HttpContext context = new BasicHttpContext(null);
        HttpHost host = new HttpHost("localhost", 50100);

        DefaultHttpClientConnection conn = new DefaultHttpClientConnection();
        ConnectionReuseStrategy connStrategy = new DefaultConnectionReuseStrategy();

        context.setAttribute(ExecutionContext.HTTP_CONNECTION, conn);
        context.setAttribute(ExecutionContext.HTTP_TARGET_HOST, host);

        try {

            String[] targets = {
                "/",
                "/servlets-examples/servlet/RequestInfoExample",
                "/somewhere%20in%20pampa"};

            for (int i = 0; i < targets.length; i++) {
                if (!conn.isOpen()) {
                    Socket socket = new Socket(host.getHostName(), host.getPort());

                    conn.bind(socket, params);
                }
                BasicHttpRequest request = new BasicHttpRequest("GET", targets[i]);
                System.out.println(">> Request URI: " + request.getRequestLine().getUri());
            }
        }
    }
}

```

```

        context.setAttribute(ExecutionContext.HTTP_REQUEST, request);
        request.setParams(params);
        httpexecutor.preProcess(request, httpproc, context);
        HttpResponse response = httpexecutor.execute(request, conn,
context);

        httpexecutor.postProcess(response, httpproc, context);

        System.out.println("<< Response: " +
response.getStatusLine());
        System.out.println(EntityUtils.toString(response.getEntity())
);

        System.out.println("=====");
        if (!connStrategy.keepAlive(response, context)) {
            conn.close();
        } else {
            System.out.println("Connection kept alive...");
        }
    }
} finally {
    conn.close();
}
}
}
}

```

