

Multitarefa



Iterativo

- Um servidor que atende a apenas um cliente por vez
 - Trabalho sequencial.
 - Finaliza o atendimento de um cliente para poder atender ao próximo
- Útil para clientes que requerem pequenas e limitadas quantidades de dados
 - Em serviços longos, a espera pode ser inaceitável



Multitarefa

- A idéia é que cada conexão possa ser processada de forma independente
 - Uma conexão trabalha sem interferir nas demais
- Threads
 - Aplicações concorrentes
 - Abstrações
 - Designer das aplicações



Threads

- Idéia:
 - Processadores que realizam múltiplas instruções
- Programação concorrente:
 - Processos;
 - Threads
- Em grande parte das implementações processos e threads compartilham o tempo de execução de um core



Processos x Threads

- Processos
 - Têm ambiente de execução próprio
 - Espaço próprio de memória de dados
 - Aplicações ou programas
- Threads
 - Existem em um processo (pelo menos uma)
 - Compartilham os recursos de um mesmo processo
 - Processos leves



Threads

```
public class GeraPDF {  
    public void rodar () {  
        // lógica para gerar o pdf...  
    }  
}  
  
public class BarraDeProgresso {  
    public void rodar () {  
        // mostra barra de progresso e vai atualizando ela...  
    }  
}
```



Threads

```
public class MeuPrograma {  
    public static void main (String[] args) {  
  
        GeraPDF gerapdf = new GeraPDF();  
        Thread threadDoPdf = new Thread(gerapdf);  
        threadDoPdf.start();  
  
        BarraDeProgresso barraDeProgresso = new BarraDeProgresso();  
        Thread threadDaBarra = new Thread(barraDeProgresso);  
        threadDaBarra.start();  
  
    }  
}
```



Java Threads

- Existem duas estratégias básicas para usar Threads para criar uma aplicação concorrente:
 - Estender a classe thread

```
public class HelloThread extends Thread {  
  
    public void run() {  
        System.out.println("Hello from a thread!");  
    }  
  
    public static void main(String args[]) {  
        (new HelloThread()).start();  
    }  
}
```



Estendendo Thread

- Definir uma classe que herda de Thread
- Utilizar o método run() na classe para implementar a tarefa
- A thread só começa a sua execução com o método start()



Java Threads

- Existem duas estratégias básicas para usar Threads para criar uma aplicação concorrente:
 - Implementar a Interface Runnable

```
public class HelloRunnable implements Runnable {  
  
    public void run() {  
        System.out.println("Hello from a thread!");  
    }  
  
    public static void main(String args[]) {  
        (new Thread(new HelloRunnable())).start();  
    }  
}
```



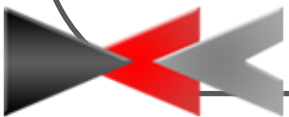
Interface Runnable

- Definir uma classe que implementa a interface Runnable
- Utilizar o método run() na classe para implementar a tarefa
- Essa classe deve ser passada como parâmetro para o construtor da thread
- A thread só começa a sua execução com o método start()



Considerações de Design

- Usar a implementação do Runnable dá maior flexibilidade ao Design, pois permite que a classe que implementou a interface estenda o comportamento de outra classe



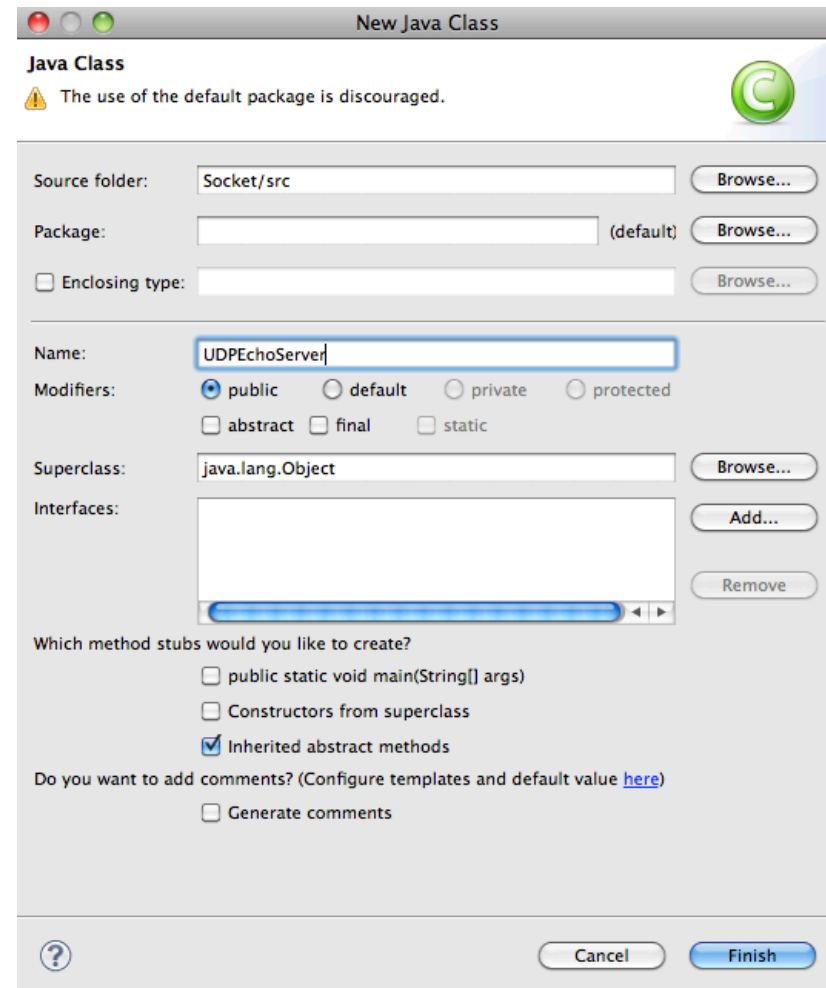
Outros Métodos

- `start()`
 - Inicia a execução do thread (só pode ser invocado uma vez)
- `yield()`
 - Faz com que a execução do thread corrente seja suspensa (e outro thread será escalonado)
- `sleep(t)`
 - Suspende o thread por t segundos
- `wait()`
 - Suspende o thread até sua reativação



ThreadExample

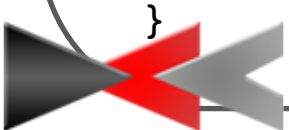
- Seleccione o Projeto
 - File
 - New
 - Class
 - ThreadExample



Exemplo

```
import java.util.concurrent.TimeUnit;

public class ThreadExample implements Runnable {
    private String sentence;
    public ThreadExample(String sentence){
        this.sentence = sentence;
    }
    public void run() {
        while(true) {
            System.out.println(sentence);
            try {
                TimeUnit.MILLISECONDS.sleep(((long)Math.random()
*100));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

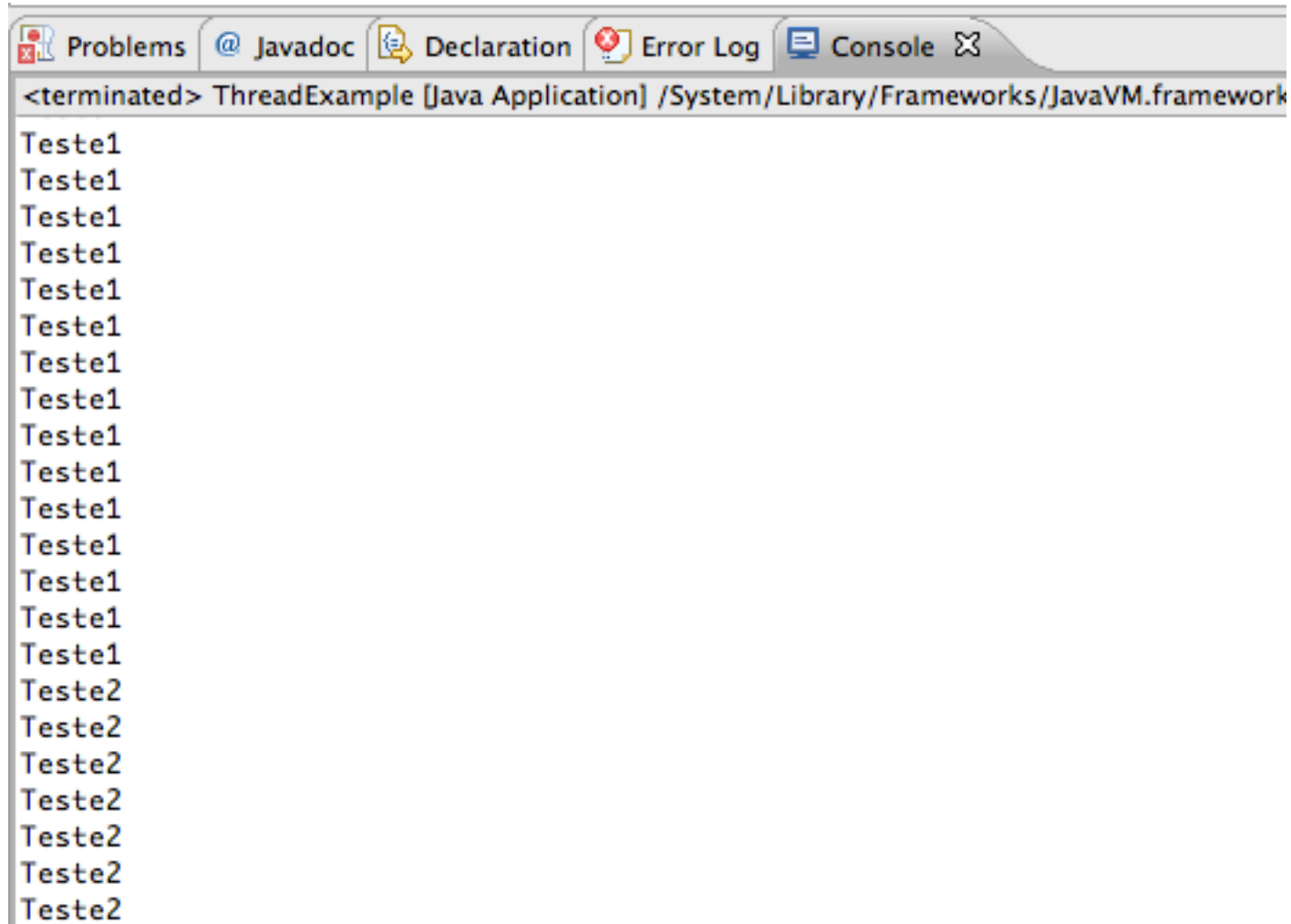


Exemplo

```
public static void main(String[] args) {  
    new Thread(new ThreadExample("Teste1")).start();  
    new Thread(new ThreadExample("Teste2")).start();  
    new Thread(new ThreadExample("Teste3")).start();  
}  
}
```



Resultado



```
<terminated> ThreadExample [Java Application] /System/Library/Frameworks/JavaVM.framework
Teste1
Teste1
Teste1
Teste1
Teste1
Teste1
Teste1
Teste1
Teste1
Teste1
Teste1
Teste1
Teste1
Teste1
Teste1
Teste1
Teste1
Teste1
Teste2
Teste2
Teste2
Teste2
Teste2
Teste2
Teste2
Teste2
```



Exemplo

- E se retirarmos o sleep?
- Resultado
 - Sequencial?
 - Alternado?
 - Randômico?



Ciclo de Vida

- E se retirarmos o sleep?
- Resultado
 - Sequencial?
 - Alternado?
 - Randômico?



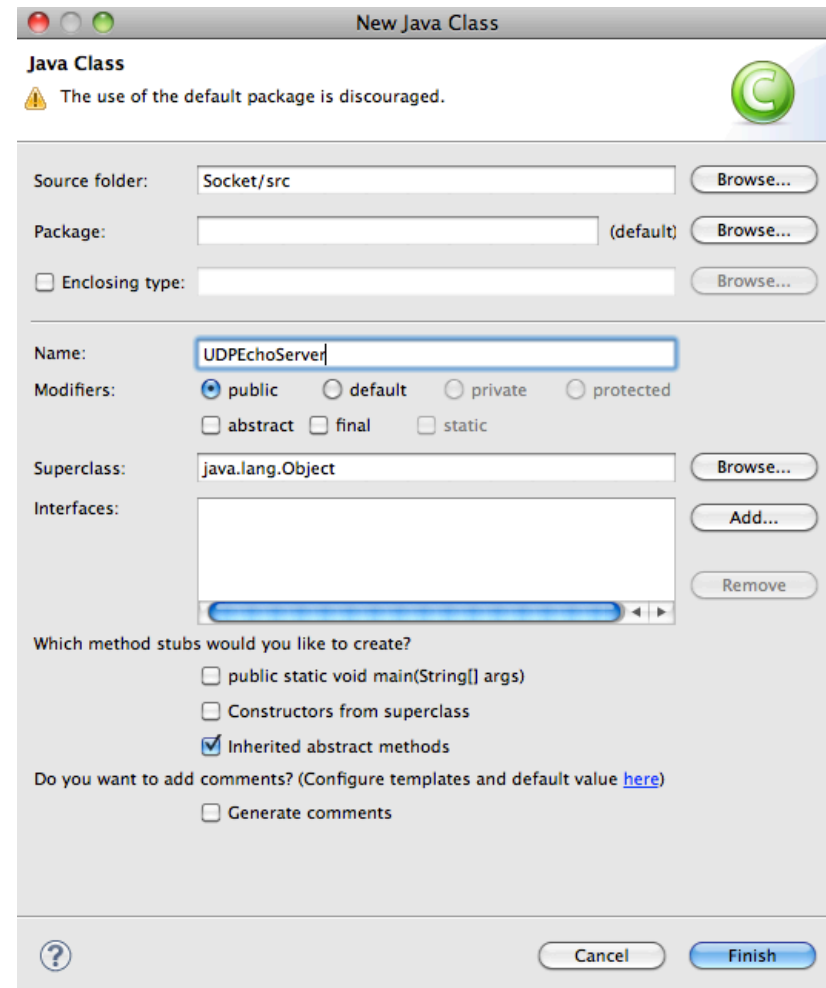
Servidor com Threads

- A idéia é construir uma thread para atender a cada cliente
- Implementar uma classe (thread) de acordo com o objetivo do servidor
 - Classe EchoProtocol
- Implementar uma classe (servidor) que recebe as solicitações e instancia as threads para tratá-las



EchoProtocol

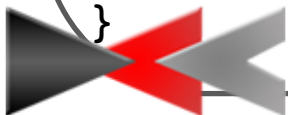
- Seleccione o Projeto
 - File
 - New
 - Class
 - EchoProtocol



EchoProtocol

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

public class EchoProtocol implements Runnable {
    private Socket clntSock;
    public EchoProtocol(Socket clntSock) {
        this.clntSock = clntSock;
    }
    public static void handleEchoClient(Socket clntSock)
                                   throws IOException {...}
    public void run () {
        try {
            handleEchoClient(clntSock);
        } catch (IOException e) {
            e.printStackTrace(); };
    }
}
```



EchoProtocol

```
public static void handleEchoClient(Socket clntSock)
                                throws IOException {

    System.out.println("Controlando cliente " +
clntSock.getInetAddress().getHostAddress() + " na porta
" + clntSock.getPort());

    InputStream in = clntSock.getInputStream();
    OutputStream out = clntSock.getOutputStream();
    int recvMsgSize;
    byte[] byteBuffer = new byte[255];
    while ((recvMsgSize = in.read(byteBuffer)) != -1)
        out.write(byteBuffer, 0, recvMsgSize);

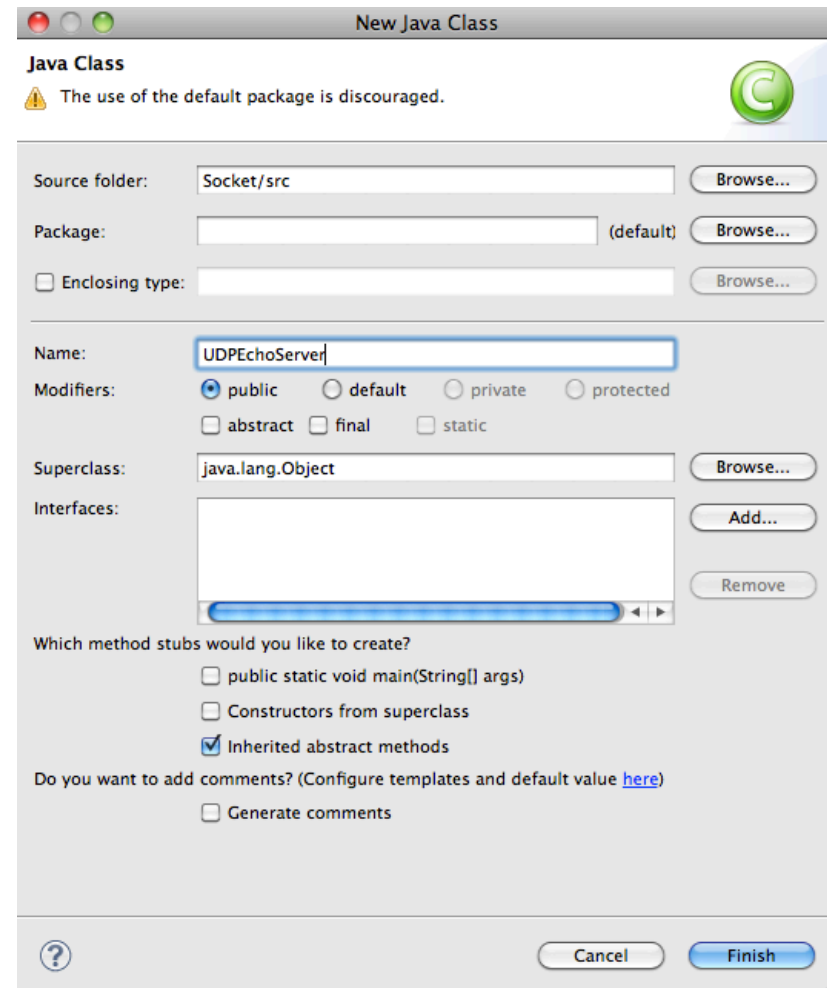
    clntSock.close();
```

}



TCPEchoServerThread

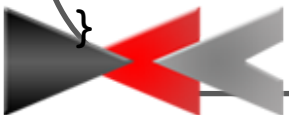
- Seleccione o Projeto
 - File
 - New
 - Class
 - TCPEchoServerThread



TCPEchoServerThread

```
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class TCPEchoServerThread {
    public static void main(String[] args) throws
        IOException {
        int echoServPort = Integer.parseInt(args[0]);
        ServerSocket servSock = new ServerSocket(echoServPort);
        while (true) {
            Socket clntSock = servSock.accept();
            Thread thread = new Thread(new
                EchoProtocol(clntSock));
            thread.start();
        }
    }
}
```



Um pouco mais sobre Threads

- Cada thread consome recursos do sistema
 - Ciclos de CPU
 - Memória
 - Etc.
- Trocas de contexto
 - Cada vez que uma thread é interrompida e outra iniciada
- O sistema pode estar gastando mais tempo gerenciando as threads do que no trabalho que ele deve fazer



Soluções

- Limitar o número de threads
 - Reusando as threads ao invés de instanciar uma nova thread para cada conexão
- Thread Pool
 - Número fixo de threads
 - Novos clientes são associados a um thread
 - Quando a conexão termina a thread está disponível para outros clientes
 - Quando as threads estão ocupadas os clientes permanecem aguardando em uma fila



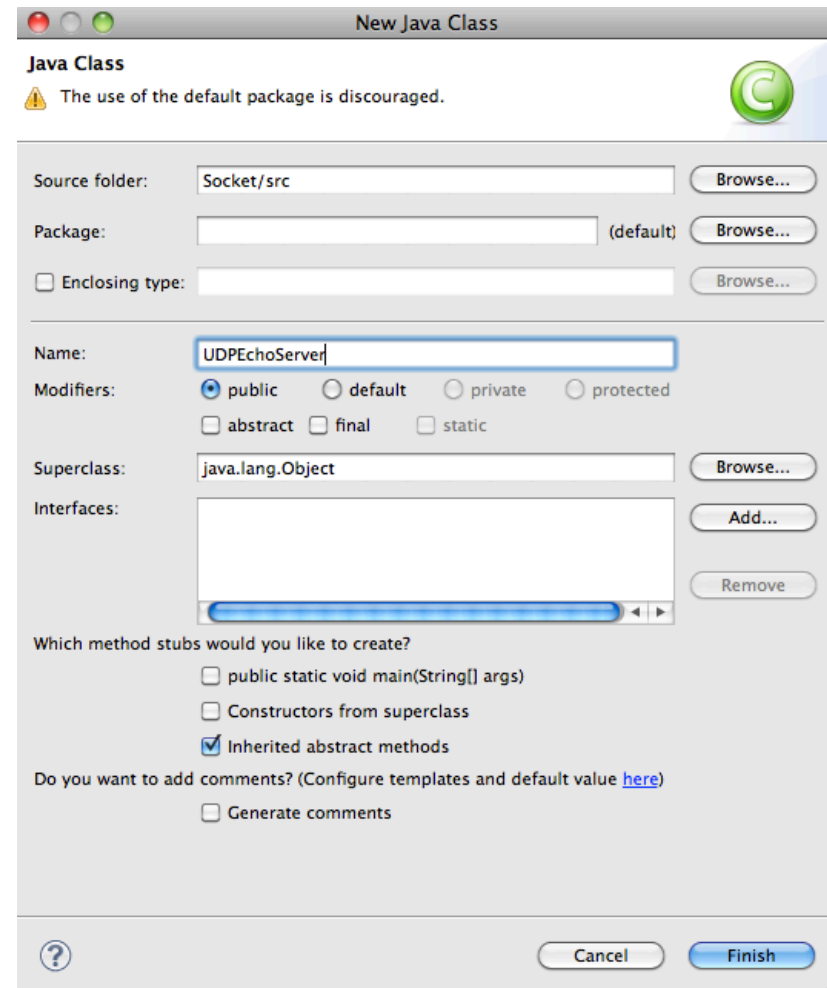
Thread Pool

1. Cria um Server Socket
 2. Cria N threads, cada uma aceitando conexões
- Controlando o número de threads, é possível distribuir os recursos do sistema
 - Se existirem poucas threads, os clientes podem permanecer aguardando um tempo grande
 - O ideal seria poder escalonar a quantidade de threads



TCPEchoServerPool

- Seleccione o Projeto
 - File
 - New
 - Class
 - TCPEchoServerPool



TCPEchoServerPool

```
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class TCPEchoServerPool {
    public static void main(String[] args) throws
        IOException{

        int echoServPort = Integer.parseInt(args[0]);
        int threadPoolSize = Integer.parseInt(args[1]);

        final ServerSocket servSock = new ServerSocket
            (echoServPort);
```



TCPEchoServerPool

```
for (int i=0; i < threadPoolSize; i++) {  
    Thread thread = new Thread() {  
        public void run() {  
            while(true) {  
                try {  
                    Socket clntSock = servSock.accept();  
                    EchoProtocol.handleEchoClient(clntSock);  
                } catch (IOException e) {  
                    e.printStackTrace();  
                }  
            }  
        }  
    };  
    thread.start();  
    System.out.println("Instanciada a thread " +  
thread.getName());  
}  
}
```

