UNIVERSIDADE PAULISTA

B22816-4 MARCIO FERNANDES CRUZ

B56734-1 DIEGO DA SILVA SANTANA

B50941-4 RAFAEL DE OLIVEIRA MARINHO

990535-9 FELIPE APOLINARIO

“Desenvolvimento de uma ferramenta para a comunicação em rede”

SÃO PAULO

2014

B22816-4 MARCIO FERNANDES CRUZ
B56734-1 DIEGO DA SILVA SANTANA
B50941-4 RAFAEL DE OLIVEIRA MARINHO
990535-9 FELIPE APOLINARIO

“Desenvolvimento de uma ferramenta para a comunicação em rede”

Dados obtidos através dos métodos e aplicações estudados em aulas e com pesquisas para fins de obtenção de nota para o curso de Ciência da Computação 4/5° Semestre (Noturno) da Universidade Paulista (UNIP) sendo entrega ao Professor/Orientador.

Professor Caruso.

Ciência da Computação 4° e 5º Semestre – Noturno

SÃO PAULO
2014

Guia de normalização para apresentação de trabalhos acadêmicos da Universidade Paulista. / Ana Lúcia E. Pires... [et al]. – São Paulo, 2012

OBJETIVO DO TRABALHO

Este trabalho tem como objetivo elaborar as Atividades Práticas Supervisionadas (APS), sendo criado um projeto de software para desenvolver um programa para catalogar e mapear as espécies de fauna que vivem a margem do Rio Tietê.

O software vai ser utilizado por zoólogos que trabalharão no projeto Tietê que, desde 1992, atua com o objetivo de despoluir o Rio Tietê. É sabido que conforme o nível da poluição do rio aumenta, a fauna diminui proporcionalmente, mas até então não houve uma catalogação deste desequilíbrio.

Os dados catalogados serão informados aos responsáveis do projeto Rio Tietê, para que eles possam publicar a população em geral. O trabalho de catalogação não será feito por um profissional apenas. Nos diversos trechos do Rio, iniciando-se em Salesópolis poderá haver equipes trabalhando. Cada equipe poderá ter de um ou mais computadores portáteis e, os dados catalogados no local, bem como troca de mensagens de textos, poderão ocorrer entre as estações.

RESUMO

O trabalho apresenta o desenvolvimento de um programa de computador que terá a finalidade de catalogar espécies de animais encontrados as margens do Rio Tietê, em seus diversos trechos, bem como executar troca destas informações entre as estações de trabalho próximas.

Como os locais de trabalho, que é a margem do Rio, podem ser distantes de redes de telefonia móvel, ausência de corrente alternada de eletricidade, bem como dificuldade em transportar equipamentos como roteadores WIFI, optou-se utilizar comunicação ponto a ponto entre as estações.

Os técnicos ou zoólogos contratados pelo projeto do Rio Tietê trabalharão próximos, e como ficarão fazendo campana para catalogar os animais em vários horários do dia e da noite, não poderão fazer barulho de voz e nem de teclas do notebook, para não espantar os animais.

O programa terá uma interface simples, e haverá vários botões na tela para comunicação entre as equipes do local de trabalho. As operações podem ser feitas através de mouse, touchpad do notebook ou mesmo, para notebooks mais avançados, toque na tela.

A fim de economizar energia, bem como não gerar muita luminosidade, a tela do programa terá cores escuras, um painel de botões e uma área de notificação que exibirá as ações de catalogação feitas por outros programas sendo executados em outras estações.

ABSTRACT

The paper presents the development of a computer program that will aim to catalog species of animals found on the banks of the Tietê River, in its various sections, as well as performing exchange of this information between stations next job.

As the workplace, which is the bank of the river, may be distant from mobile phone networks, absence of alternating current electricity, as well as difficulty in transporting equipment such as WIFI routers, we chose to use point to point communication between stations.

Technical or zoologists hired by the Rio Tietê work closely , and how will doing bell to catalog the animals at various times of day and night , shall not make any noise and voice nor the notebook keys , not to scare away the animals .

The program will have a simple interface, and there will be several buttons on the screen for communication between the teams in the workplace. Transactions can be made via mouse, notebook touchpad or even for more advanced notebooks, tap the screen.

In order to save energy, and do not generate a lot of brightness, the screen of the program will have dark colors, a panel of buttons and a notification area that displays the actions of cataloging done by other programs running in other seasons.

SUMÁRIO

Lista de figuras

Introdução

No ano de 1992, ocorreu no Rio de Janeiro a CNUMAD (Conferência das Nações Unidas sobre o Meio Ambiente). Esta conferência internacional fez mobilizar várias campanhas no Brasil e no mundo com a preocupação com o meio ambiente. Neste ano, o governo do Estado de São Paulo, junto com a ONG SOS Mata Atlântica conseguiu juntar fazer um abaixo assinado com mais de um milhão de assinaturas para se criar um projeto para despoluir o Rio Tietê. Este projeto se chama Projeto Tietê Vivo.

Este projeto foi criado pela Sabesp, tendo o seu objetivo de melhorar a qualidade da água de córregos e rios, e principalmente para o Rio Tietê. A Sabesp já executou duas etapas desse projeto, e atualmente está executando a terceira etapa, com o objetivo de ultrapassar as metas obtidas anteriormente na coleta e no tratamento.

Estão sendo mais de 500 empreendimentos espalhado na grande São Paulo, para construir ou ampliar estações de tratamento de esgoto, instalar novas tubulações, aumentar as redes coletoras nos bairros e nova ligação domiciliares. Todo o esgoto gerado é encaminhado ao tratamento, e é feito todo o processo de despoluição e devolvido a natureza.

Com esse projeto, já foi percebido que nas cidades dos interiores, tem visto a volta dos peixes e a redução da mancha de poluição. A despoluição dos rios na grande São Paulo e Rio Tietê, dependem muito da população, colaborando a evitar danos que contribui para a poluição.

Uma das preocupações do Projeto Tietê vivo é a preservação da fauna que vive e depende de um rio despoluído para se preservar. Geralmente se faz estudos para saber a quantidade e a diversidade de animais que vivem em diversos trechos do Rio, sejam eles poluídos e despoluídos. Segundo o site Portal do Departamento de Águas e Energia Elétrica[1], a poluição já se inicia a 45 km, no município de Mogi das Cruzes e há necessidade de se obter índices de poluição e da fauna em todos os trechos.

[1] (http://www.daee.sp.gov.br/index.php?option=com_content&view=article&id=793:historico-do-rio-tiete&catid=48:noticias&Itemid=53).

Os especialistas do projeto Tietê necessitam de dados catalogados nos diversos trechos do Rio e, estas informações, só são possíveis se obter in loco. E sabido e a suposições de um número que se possa definir entre o aumento de índices de poluição e deterioração da fauna. Para isso, faz-se necessário o desenvolvimento de um software que catalogue estas informações, possa ser rápido para catalogar, seja discreta para não fazer barulho ou trazer alta luminosidade à tela para preservar o consumo de energia de computadores portáteis e, por último e mais importante, possa possibilitar a troca de mensagens entre os usuários.

Os locais de estudo do Rio podem ser áreas longe de redes de telefonia móvel, bem como não podem usar WIFI, pois muitas vezes fica impraticável ligar um roteador a uma tomada de energia, mesmo com um no-break. A utilização de muito equipamento pode dificultar o trabalho dos técnicos e zoólogos do projeto Tietê Vivo.

O programa que vai rodar em estações terão 2 funções básicas: catalogar os animais conhecidos, como capivaras, tatus, cobras e, enviar mensagens as outras estações de trabalho. Para cada catalogação de animal será aberto uma caixa de diálogo, que o usuário, opcionalmente poderá informar dados mais objetivos da catalogação e, esta observação adicional é registrada na base de dados do determinado programa e também é transmitida como dado catalogado as outras máquinas da rede, através do servidor. Todos os dados catalogados poderão ser exportados para uma planilha eletrônica, possibilitando os responsáveis do projeto Bichos do Tietê tomar decisões em função dos dados catalogados.

1. Fundamentos de comunicação de dados em redes

Uma rede de computadores é formada de no mínimo dois computadores interligados, podendo haver ou não compartilhamento de recursos entre eles. A criação das redes foi motivada para promover a troca de informações e compartilhamento de recursos então, há sempre algum nível de ligação, seja num menor ou maior grau, senão, não teria sentido uma estação de trabalho estar isolada, fora da rede, por exemplo.

Uma das grandes vantagens que uma rede de computadores proporciona é a economia de recursos e de tempo de trabalho ao se compartilhar periféricos. Um exemplo conhecido pelas empresas é a economia que se ganha com as impressoras. Estes hardwares consomem papel, tinta e gera alto custo de manutenção. É visivelmente uma despesa.

O custo para se mantiver impressoras é multiplicado pelo número destas dentro de uma organização. Graças às redes e sua capacidade de prover compartilhamento de recursos, reduz-se o número de periféricos e, assim, o custo operacional para mantê-los. Geralmente o compartilhamento de recursos de hardware numa rede ocorre em redes locais, mas, existem tecnologias hoje que permitem a utilização de dispositivos entre redes.

Um exemplo disso é a impressão remota utilizada por alguns aplicativos como o Google Cloud Print2. Outra utilização importante de redes é o compartilhamento de dados e informações. Isto engloba desde compartilhamento de planilhas e documentos entre usuários até a utilização de sistemas gerenciais, não importando se isso é feito numa mesma rede local ou numa rede geograficamente distante.

Empresas matrizes e suas filiais mantêm sistemas 2interligados e troca de arquivos com ligações entre redes de forma transparente ao usuário final. Possivelmente estas redes conectadas não provêm taxas próximas a 100% de disponibilidade por fatores externos, pois pode estar em manutenção o serviço

[2] (http://www.google.com.br/cloudprint/learn/apps.html).

de telefonia ou mesmo a antena micro-ondas ter sido danificada, mas, em tese, atende muito bem as empresas.

2. Topologias de Redes

Num modo geral, as redes de computadores estão presentes em nosso dia a dia, por conta do costume ao utilizá-las, não damos conta da sofisticação e complexidade desta estrutura que faz com que os dados e informações ao nosso redor transmitem. Devemos levar em consideração como é importante a maneira com que as redes de computadores são interligadas, pois podemos conectar dispositivos de várias formas, envolvendo tanto a parte física quanto a parte lógica.

A diferença é que na Topologia Física nos referiu ao layout físico aos meios de conexão dos dispositivos de redes, ou seja, como eles são realmente conectados. Os dispositivos que fazem parte da estrutura de uma rede recebem o nome de nós ou nodos. Já na Topologia Lógica, é a maneira como os nós fazem a comunicação através dos meios de transmissão. As redes de computadores são compostas por arranjos topológicos interligados e, sua principal finalidade é economizar recursos, pois com suas estruturas, compartilhamento e processamento individual são distribuídos para todos, fazendo com que as informações cheguem a todos os usuários que estão conectados.

2.1. Topologias Físicas

Existem 8 tipos de topologia físicas, são elas:

- Ponto a Ponto: Consideramos a mais simples, onde pode ser representada por apenas dois computadores que são interligados entre si, utilizando um meio de transmissão qualquer, e também serve como base para formar novas topologias, incluindo outros nós em sua estrutura.

- Barramento: O meio de transmissão irá ligar todos os nós, os nós que estiverem ligados à barra poderão ouvir as informações que estão sendo transmitidas. Existem dois modos de controle para que se possa ser feito o

controle ao acesso das estações do barramento, o centralizado, que é um nó especial que permite ou não o direito de outro nó acessar o barramento e, o descentralizado onde o controle de acesso é distribuído entre os nós.

- Anel: A topologia em Anel é formada por nós conectados através de um circuito fechado, onde o sinal passa de estação em estação até chegar ao seu destino, neste caso as estações fazem o papel de repetidores e retransmitem o sinal recebido até que o destinatário seja encontrado e receba a informação.

- Estrela: É formada por um dispositivo central que liga todas as demais estações e a comunicação é toda supervisionada por este nó central. Se houver falhas nas estações ou na ligação entre a estação e o dispositivo central, apenas o nó que estiver envolvido na ligação ficará de fora, porém se houver uma falha no dispositivo ou nó central, todas o sistema ficará fora do ar.

- Árvore: É como se tivéssemos várias redes estrelas interligadas entre si. Geralmente existe uma barra central onde outros ramos menores se conectam. Devemos tomar alguns cuidados neste tipo de Topologia, pois cada ramificação significa que o sinal deve se propagar por caminhos distintos.

- Híbrida: É a mais utilizada em grandes redes, por utilizar mais de uma Topologia ao mesmo tempo e, suas características são as ligações ponto a ponto e multi ponto, obtendo-se com isto redes complexas proporcionando um maior número de recurso. Várias configurações podem ser criadas utilizando uma variação de outras Topologias.

- Lógicas: As Topologias Lógicas significam a forma com que os nós fazem a comunicação através dos meios físicos, existem dois tipos mais comuns de Topologias Lógicas que são o Broadcast e a Passagem de Token. Na Topologia de Broadcast, o nó envia suas informações a todos os outros nós da rede, não há uma ordem para o envio destas informações, apenas que o primeiro a chegar, é o primeiro a usar. Teremos a Ethernet como exemplo deste funcionamento abaixo:

- Token Ring: Este método utiliza a topologia em Anel para que seja transmitido dado entre duas estações. A estação que quer transmitir os dados,

precisa obter um sinal (Token), que permite a estação o direito de transmitir e fazer o percurso de nó em nó. Como se tem apenas um Token disponível na rede, não há colisão de pacotes, já que uma única estação irá acessar a rede por vez.

3. Tipos de transmissão

3.1. Sinais elétricos

Podemos definir como tensões, porém existem vários tipos, mas alguma delas é uteis, por transmitirem dados que trafegam nas redes de computadores. Essas tensões podem ser classificadas em dois tipos que são sinais analógicos e digitais.

Parte dos sinais elétricos utilizado na computação são os sinais digitais, e os analógicos primeiramente são digitalizados para depois serem processados e armazenados. Os sinais digitais são uma infinidade de valores, mas porem matematicamente eles não são perfeitos por apresentarem mais de dois valores, e o necessário seria apresentar somente dois valores 0 e 1.

3.2. Modo de operação

Em qualquer tipo de comunicação, o envio e o recebimento de dados podem ou não existir simultaneamente e estão classificados em vários tipos. São eles:

-Simplex: A comunicação só poderá ser realizada em uma única direção.

- Half-Duplex: A comunicação pode ser realizada em ambas as direções, porém não simultaneamente.

-Full-Duplex: A comunicação é realizada em ambas as direções simultaneamente.

3.3. Transmissões paralela e serial

A transmissão paralela é a ocorrência simultaneamente de bits, através de fios independentes, porém seu custo é mais caro devido exigir cabos complexos com vários condutores para torna sensível as interferências eletromagnéticas. Porém é mais rápida que a serial. Já a serial, o seu envio de bits, é feito um por vez, e com esse envio é possível atingir distância maiores. E o seu custo é mais barato, devido usar cabos mais simples que são mais baratos.

3.4 Ritmos de transmissão

Transmissão assíncrona é a irregularidade de ocorrência da transmissão, e nela é inserido no inicio e no fim um bit de um caractere, para que o receptor conseguir entender o que foi feito transmitido. Porém tendo uma desvantagem que é a má utilização do canal, por os caracteres não serem bem transmitido. Na transmissão síncrona os bits são rapidamente enviado um após o outro e não é necessário ter o controle de bits no inicio e no fim. Porém é mais cara que a assíncrona, por necessita de um relógio no hardware para liberar o seu sincronismo, e são bastante utilizada em redes que tem a sua taxa de transmissão alta.

O usuário final não precisa conhecer os detalhes internos do funcionamento de uma rede, mas, é obrigação do estudante de ciência da computação conhecer o funcionamento interno de uma rede. Internamente a rede é dividida em camadas. Abaixo descreveremos as característica da camada física.

4. Camada Física

É a camada mais baixa na hierarquia de uma rede. Segundo TANENBAUM [2003], ela define as interfaces mecânicas, elétricas e de sincronização de rede. São nela que se está os meios de transmissão. A base teórica da comunicação de dados é fazer-se variar e detectar esta variação de

alguma propriedade física, como a voltagem (tensão elétrica) ou a corrente. Quase se representa o valor propriedade física como uma função de tempo com um valor único, f(t), cria-se um modelo de comportamento de sinal e é possível analisá-lo matematicamente e, uma técnica que foi criada, se chama análise de Fourier.

Um termo muito utilizado entre os meios de transmissão é a largura da banda. TANEMBAUM [2003] a define como uma propriedade física, e em geral depende da construção, da espessura e do comprimento do meio de transmissão. Ainda segundo este autor, nenhum recurso de transmissão é possível transmitir sem perder parte da energia no processo. Em alguns casos é introduzido um filtro para limitar o volume de largura de uma banda para cada usuário final. Teoricamente uma linha telefônica comum pode ter uma largura de banda de 1MHz para curtas distâncias mas, as empresas de telefonia, segundo TANEMBAUM[2003] acrescentam um filtro e restringe para a 3100 MHz. Pois, este número é adequado para voz e ao mesmo tempo melhora a eficiência do sistema como um todo, ou seja, todos os clientes não perdem.

Voltando para a definição dos meios de transmissão, será descrito abaixo os meios de transmissão.

4.1. Par Trançado

Segundo TANEMBAUM [2003], a aplicação mais comum do par trançado é o sistema telefônico. Quase todos os telefones estão conectados à estação central da companhia telefônica por um par trançado. Os pares trançados podem se estender por diversos quilômetros sem amplificação, mas, quando se trata de distâncias mais longas, existe a necessidade de repetidores. Quando muitos pares trançados percorrem paralelamente uma distância muito grande, como acontece na ligação entre um prédio e a estação central da companhia telefônica, eles são envolvidos por uma capa protetora. Se não estivessem trançados, esses pares provocariam muitas interferências. Nos países em que as linhas telefônicas são instaladas em postes, com frequência vemos cabos de pares trançados com vários centímetros de diâmetro. Ainda segundo TANENBAUM [2003], os pares trançados podem ser

usados na transmissão de sinais analógicos ou digitais. A largura de banda depende da espessura do fio e da distância percorrida, mas, em muitos casos, é possível alcançar diversos megabits/s por alguns quilômetros. Devido ao custo e ao desempenho obtidos, os pares trançados são usados em larga escala e é provável que assim permaneçam nos próximos anos.

4.2. Cabo Coaxial

De acordo com TANEMBUAM [2003], tem melhor blindagem que os pares trançados, e assim pode se estender por distâncias mais longas em velocidades mais altas. Dois tipos de cabo coaxial são amplamente utilizados. O grosso e o fino. O primeiro tem alcance máximo de 510 metros sem repetidor e tem a desvantagem de ser caro. O fino é muito mais barato que o grosso e o alcance máximo é 170m.

Ainda segundo TANENBUAM [2003], um cabo coaxial consiste em um fio de cobre esticado na parte central, envolvido por um material isolante. O isolante é protegido por um condutor cilíndrico, geralmente uma malha sólida entrelaçada. O condutor externo é coberto por uma camada plástica protetora. A construção e a blindagem do cabo coaxial proporcionam a ele uma boa combinação de alta largura de banda e excelente imunidade a ruído.

4.3. Fibra óptica

Segundo TANEMBAUM [2003], um sistema de transmissão óptica tem três componentes fundamentais: a fonte de luz, o meio de transmissão e o detector. Por convenção, um pulso de luz indica um bit 1, e a ausência de luz representa um bit zero. O meio de transmissão é uma fibra de vidro ultrafina. O detector gera um pulso elétrico quando entra em contato com a luz. Quando instalamos uma fonte de luz em uma extremidade de uma fibra óptica e um detector na outra, temos um sistema de transmissão de dados unidirecional que aceita um sinal elétrico, converte o sinal e o transmite por pulsos de luz; depois, na extremidade de recepção, a saída é reconvertida em um sinal elétrico.

Esse sistema de transmissão desperdiçaria luz e na prática não teria a menor utilidade, exceto como um interessante princípio físico. Quando um raio de luz passa de um meio para outro, por exemplo: de sílica fundida para o ar, o raio é refratado (desviado) na fronteira sílica/ar Para ângulos de incidência que ultrapassam certo valor crítico, a luz é refletida de volta para a sílica; nada escapa para o ar. Dessa forma, um raio de luz incidente no ângulo crítico ou acima dele é interceptado no interior da fibra.

4.4. Transmissões de Rádio

Segundo TANENBAUM, a ondas de rádio são fáceis de gerar, podem percorrer longas distâncias e penetrar facilmente nos prédios; portanto, são amplamente utilizadas para comunicação, seja em ambientes fechados ou abertos. As ondas de rádio também são omnidirecionais, o que significa que elas viajam em todas as direções a partir da fonte; desse modo, o transmissor e o receptor não precisam estar fisicamente alinhados.

A desvantagem do rádio é que é sujeito a interferência de equipamentos elétricos. Em baixa frequência, atravessa bem os obstáculos, mas a potência cai rapidamente e acompanha a curvatura da Terra. Em alta frequência, propaga-se praticamente em linha reta, refletindo na ionosfera.

4.5. Transmissões em Micro-ondas

Segundo TANEMBAUM, Acima de 100 MHz, as ondas trafegam praticamente em linha reta e, portanto, podem ser concentradas em uma faixa estreita. A concentração de toda a energia em um pequeno feixe através de uma antena parabólica (como a conhecida antena de TV por satélite) oferece uma relação sinal/ruído muito mais alta, mas as antenas de transmissão e recepção devem estar alinhadas com o máximo de precisão.

É utilizado principalmente em enlaces entre telefonia. Substitui o cabo coaxial e a fibra óptica para longas distâncias, mas requerem linha visada. Outra aplicação é para fazer interligação de redes (inter-redes) entre prédios.

5. Plano de desenvolvimento da aplicação

A primeira questão a ser avaliada antes de ser traçado o caminho para o desenvolvimento do aplicativo foi traçar o cenário do aplicativo dentro do contexto para saber quais as atividades industriais que estão afetando na poluição do Rio Tietê que o trabalho estava sendo exigido.

Foi traçado um plano em criar um aplicativo para catalogar e mapear as espécies de fauna que vivem a margem do Rio Tietê. A principal ideia desse aplicativo que serão usados por zoólogos é que eles possam se comunicar através de uma rede sem fio, para transmitir e receber informações entre os usuários que utilizarão essa ferramenta.

Devido o projeto criado desde 1992 que tem como objetivo de despoluir o Rio Tietê, os zoólogos trabalharão nas margens do Rio Tietê, mas eles estarão longe de telefonia móvel, ausência de corrente alternada de eletricidade, bem como dificuldade em transportar equipamentos como roteadores WIFI, optou-se utilizar comunicação ponto a ponto entre as estações.

Os zoólogos contratados pelo projeto do Rio Tietê trabalharão em vários horários de dia e de noite e não poderão estar fazendo barulho de teclas do notebook e de voz, para não espantar os animais que estão às margens do Rio Tietê. As operações podem ser feitas através de mouse, touchpad do notebook ou mesmo, para notebooks mais avançados, que utilizam a tecnologia de touch screen.

A fim de economizar energia, bem como não gerar muita luminosidade, a tela do programa terá cores escuras, um painel de botões e uma área de notificação que exibirá as ações de catalogação feitas por outros programas sendo executados em outras estações, bem como solicitações específicas para comunicação entre a equipe, por exemplo: pedido de socorro, recolher Material e ficar alerta com perigo de animais perigosos.

5.1. API de sockets em Java

Para o desenvolvimento deste aplicativo, usaremos Sockets sob o protocolo TCP/IP. Como o programa está sendo feito em Java, será usada a API Sockets, que é uma suíte de classes que possibilita o trabalho em rede com Sockets. No API de sockets, para que dois ou mais computadores possam trocar dados, há a necessidade de usar protocolo, existem diversos tipos de protocolo, mas nesse aplicativo usaremos o protocolo TCP (*Transmission Control Protocol).*

A vantagem que temos de usar TCP, é que não é necessário criar nosso próprio protocolo, e que o TCP garante a entrega de todos os pacotes que transferimos. É comum encontra máquina com uma só conexão física, mas todas as aplicações que recebe e enviam dados, faz através da mesma conexão física, e durante a chegada dos dados, o computador consegue identificar as informações que pertencem somente àquela aplicação. Assim como uma máquina pode ser identificada com IP, a porta é a solução para identificar as aplicações em uma máquina.

5.2. Comandos do aplicativo

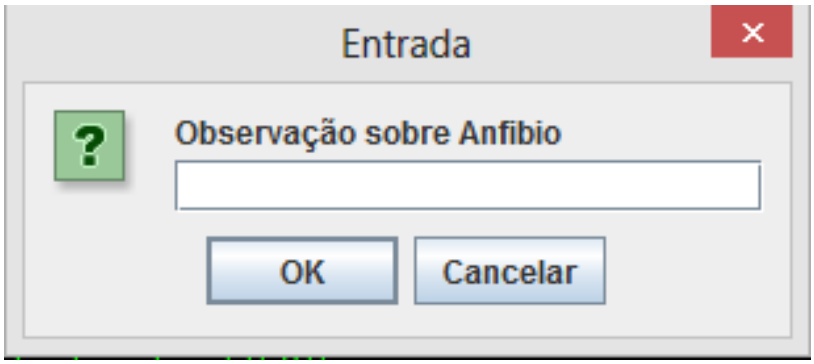
Serão descrito os comandos que os usuários usarão nessa aplicação. Na figura 1 que mostraremos abaixo, estão todos os animais que os zoólogos estarão fazendo as catalogações.

Figura 1: Animais para catalogações.

Fonte: Adaptação própria.

Ao clicarmos em um dos animais, será exibido uma tela (figura 2), onde os zoólogos poderão digitar as informações observadas dos animais, e serão listadas todas as informações com o dia da semana, data, horário e a observação feita dos animais, e todos os dispositivo que terão essa tela e tiver conectado a rede, poderão visualizar essas informações.

Figura 2: Entrada de observação.



Fonte: Adaptação própria.

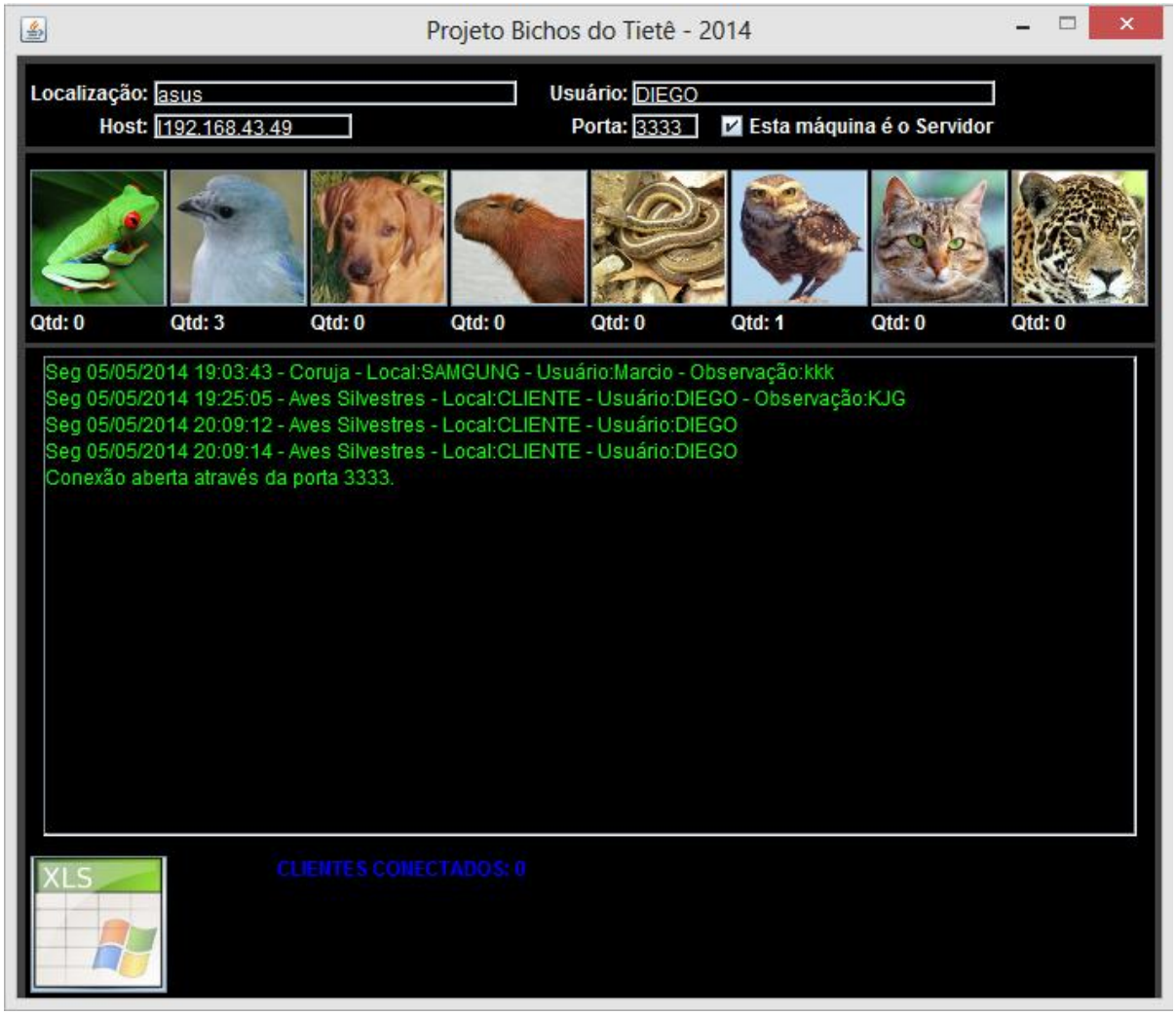
Essa figura 2 será gerada para todas as espécies de animais, para ser digitado as observações. Nesse aplicativo terá um comando que os zoólogos poderão exporta para o Excel às observações catalogadas dos animais, para que eles possam publicar a população em geral. Na figura 3 será mostrado esse comando.

Figura 3: Exportar catalogações.

Fonte: Adaptação própria.

Na Figura 4 abaixo mostraremos a Interface do programa que será utilizada pelos usuários.

Figura 4: Interface do programa com o usuário



Fonte: Adaptação própria.

5.3. Funcionamento do programa em rede

O funcionamento do aplicativo que serão usados pelos zoólogos para as catalogações dos animais, poderá ter um ou mais programas sendo rodado simultaneamente. No programa vai haver dois campos que são o host e porta, onde deve ser digitado o numero da porta e do endereço do IP, para que os usuários estejam conectados nessa mesma rede, terá que ter a tela do programa como endereço do IP e da porta idênticos para todas as maquinas.

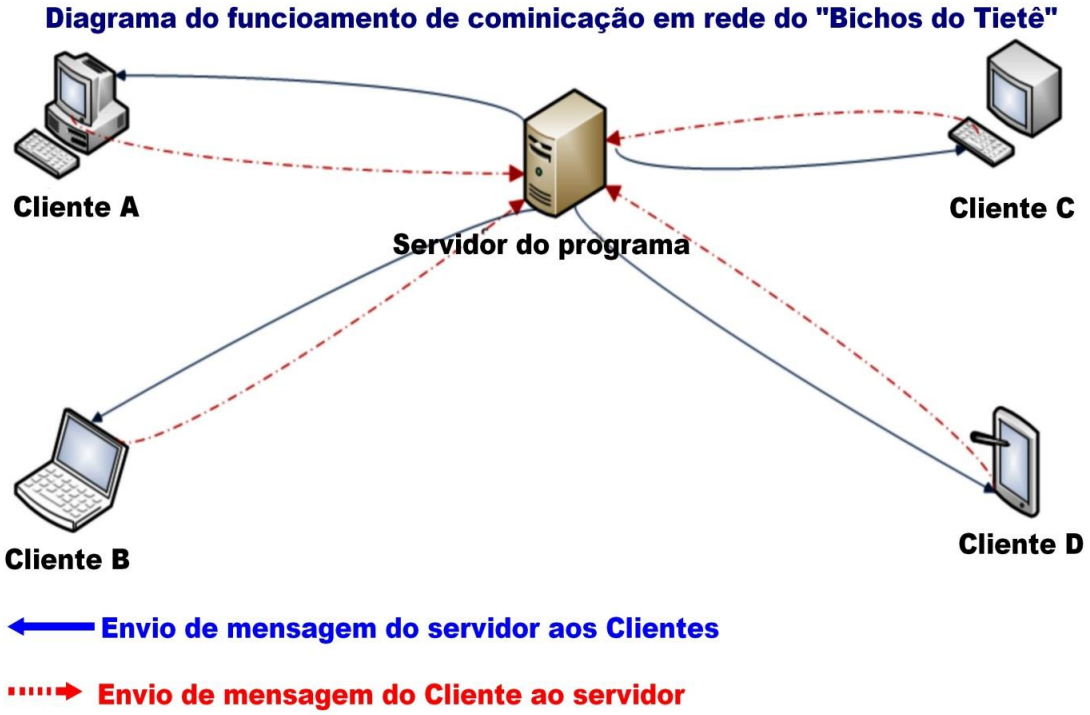
Se o usuário clicar em qualquer uma das maquinas e em qualquer um dos animais, o programa pedirá para complementar a catalogação com uma observação adicional feita do animal, ao dar entrada nessa observação, será automaticamente apontado para todas as maquinas que estiverem conectadas a rede com o mesmo endereço do IP e da porta.

Essa técnica utiliza no programa Bichos Do Tietê, e chamado Broadcasting, por definição ela é uma técnica onde transmite informações, e a sua principal característica é enviar informações para vários receptores ao mesmo tempo. Portanto nesse programa o usuário passara as informações para o servidor, onde o servidor é responsável para enviar as informações para os demais usuários conectados a esse servidor.

5.4. Diagrama do funcionamento em rede

Na figura 5, é demonstrado o funcionamento do programa, em forma de diagrama de comunicação. O Bichos do Tietê faz broadcasting de toda catalogação executada feita por qualquer programa executando em rede, através do servidor.

Figura 5: Diagrama do funcionamento



Fonte: adaptação própria.

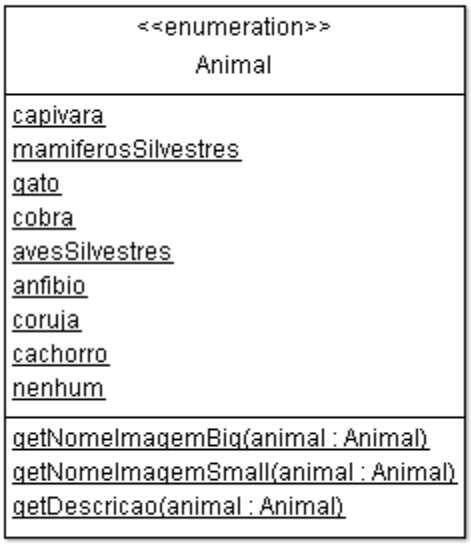
A catalogação pode ser feita em qualquer cliente e também no servidor. Quando algum programa faz a catalogação, o servidor trata de fazer Broadcasting para todos os outros programas que estão na rede.

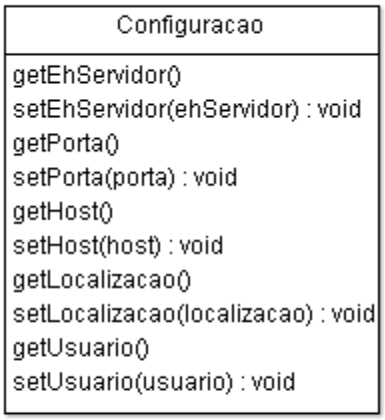
Cada programa em execução mantém um arquivo de seus próprios registros e, todos os programas tem um recurso que pode ser feito exportação dos animais catalogados para uma planilha eletrônica em formato “XLSX”.

6. Projeto, estrutura e módulos que foram desenvolvidos.

O projeto foi desenvolvido na linguagem Java e para facilitar o desenvolvido, foi dividido em packages que segue o padrão MVC (Model – View – Controller).

6.1. Package Tipos





Este package contém as estruturas de dados utilizadas no sistema e a classe com funções gerais usadas no sistema como exportação de dados para planilha Excel.

6.1.2. Classe Animais

É uma classe enumerada onde se define quais animais serão catalogados pelo programa. Possui métodos também para trazer a descrição de determinado animal, caso passamos por parâmetro do seu tipo.

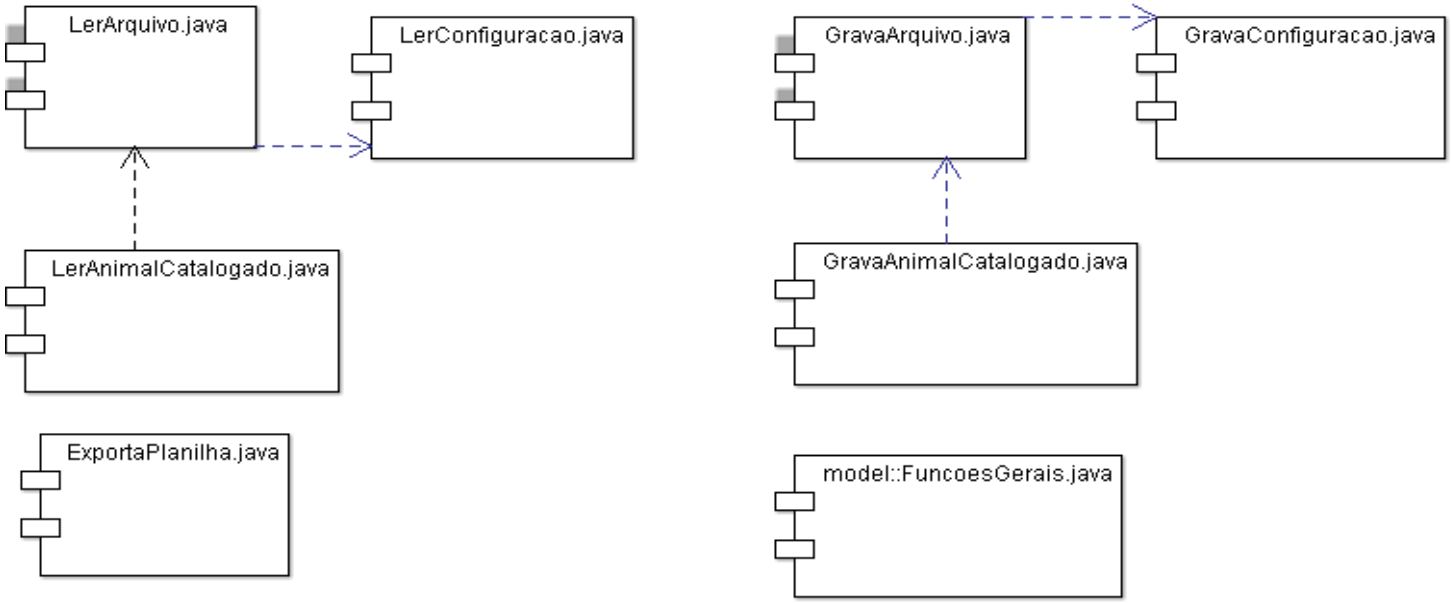
6.1.3. Classe Catalogada

É uma estrutura de dados que mantém o registro de algum animal registrado. Esta classe implementa a serialização de objetos, possibilitando persistir os dados em arquivo binário, bem como transferir mensagem via rede, através de conversão de objeto para uma cadeia de caracteres.

6.1.4. Classe Configuracao

É uma estrutura de dados que mantém a configuração de determinada estação de trabalho, basicamente gravamos o usuário e o local da catalogação de animais.

6.2. Package Model



Este pacote contém as regras de negócio do programa, de acordo com o padrão MVC.

6.2.1. Classe Funcoes Gerais

O sistema para funcionar precisa saber algumas informações em vários momentos e, estas funções como saber o caminho do banco de dados, o caminho das imagens dos bichos e função se podemos alimentar a lista de eventos estão nesta classe.

6.2.2. Classe Exporta Planilha

O sistema é uma unidade isolada de trabalho, mas seus dados devem ser exportados para ser estudados por outros aplicativos e sistemas e, a melhor forma de fazer isso é criar um recurso de exportação destes dados para uma planilha eletrônica e, esta classe é responsável por isso.

6.2.3. Classe Grava Arquivo

É a classe que contém métodos genéricos para gravação de objetos serializados no sistema.

6.2.4. Classe Grava Animal Catalogado

Esta classe estende a classe Grava Arquivo e é responsável pela gravação dos animais catalogados no disco.

6.2.5. Classe Grava Configuracao

Esta classe estende a classe Grava Arquivo e é responsável pela gravação da configuração do programa na estação do trabalho.

6.2.6. Classe Ler Arquivo

Esta classe, de forma inversa a classe Grava Arquivo, contém métodos genéricos para leitura de objetos serializados e gravados em disco.

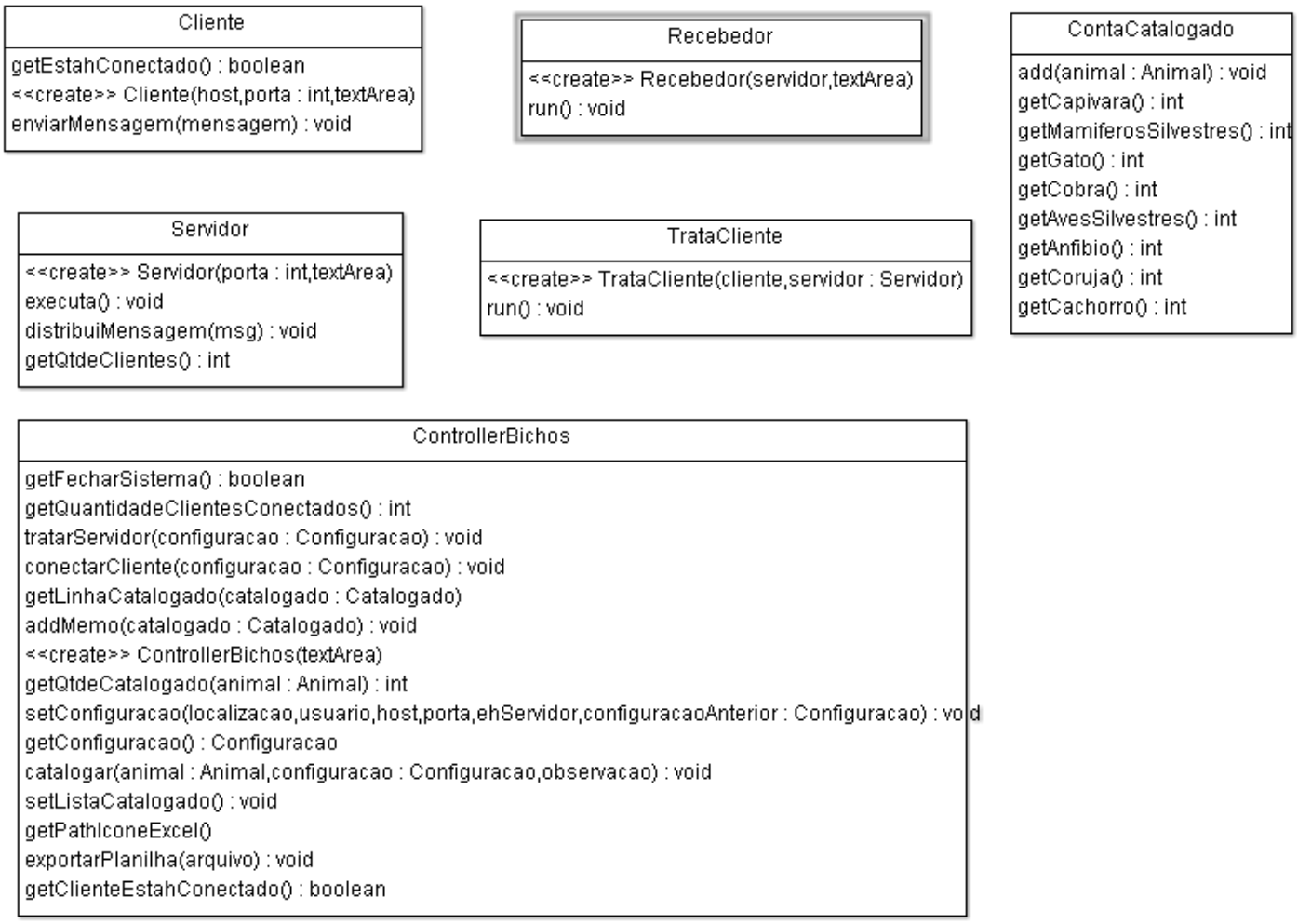
6.2.7. Classe Ler Animal Catalogado

Esta classe estende a Ler Arquivo e é responsável pela recuperação dos objetos serializados por uma gravação em um momento anterior.

6.2.8. Classe Ler Configuracao

Esta classe estende a Ler Arquivo e é responsável pela leitura da configuração previamente gravada.

6.3. Pacote Controller



Este pacote contém as classes que facilitam as chamadas ao pacote Model, que contém as regras do sistema propriamente ditas.

6.3.1. Classe Controller Bichos

Esta classe faz às chamadas as regras de negócio do programa como a gravação e exportação de dados. Esta classe é instanciada por uma classe na camada View, facilitando as chamadas sem deixar a View ter acesso direto a Model.

6.3.2. Classe Servidor

O programa Bichos do Tietê funciona em modo servidor ou modo cliente e, esta classe contém métodos que possibilita o recebimento de sockets, através de um Host e Porta previamente configurados.

6.3.3. Classe Trata Cliente

O programa Bichos do Tietê funciona em forma colaborativa entre várias estações de trabalho e, esta classe é responsável por fazer o *Broadcasting* de registros dos eventos aos vários clientes conectados ao servidor.

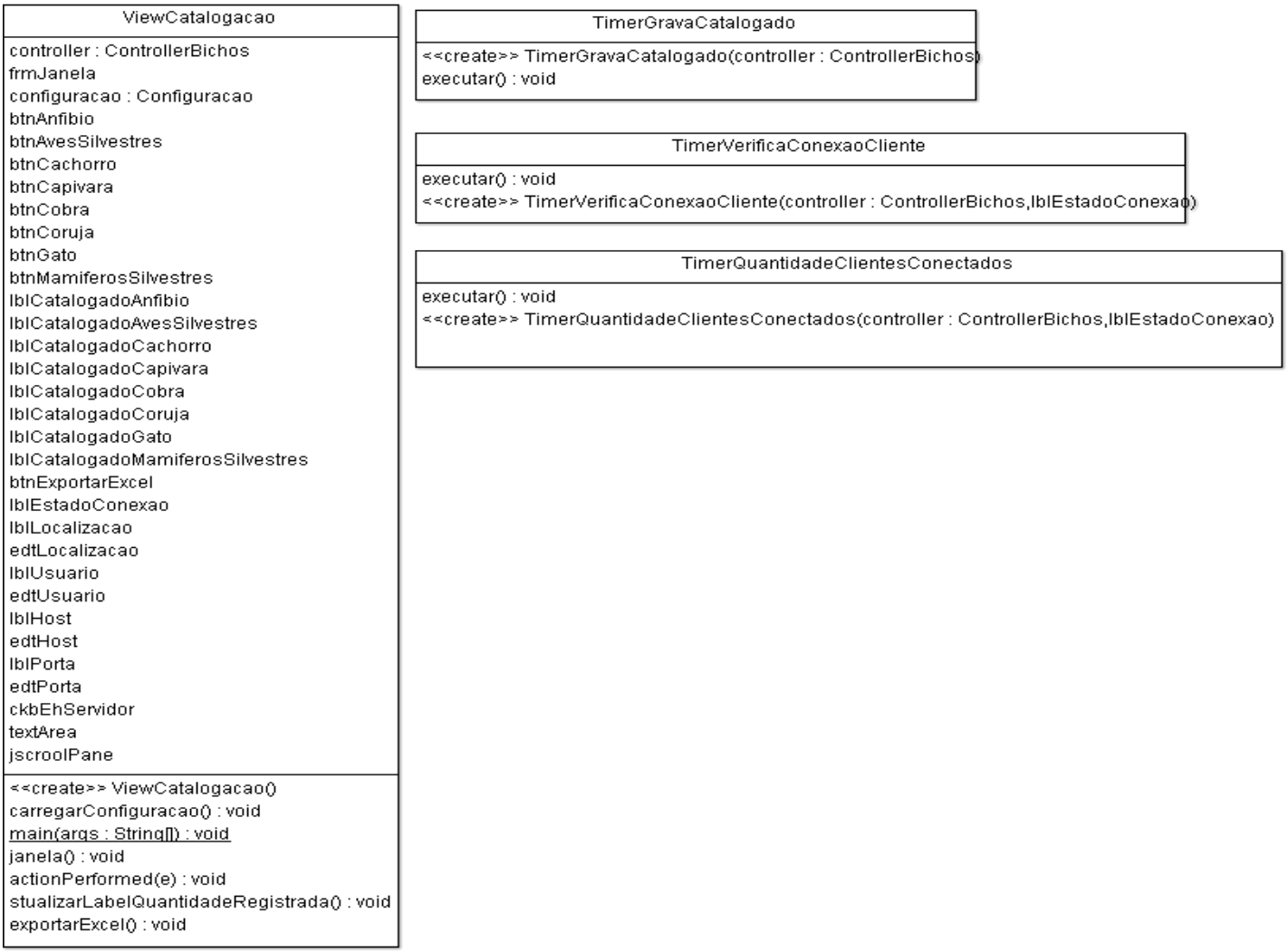
6.3.4. Classe Cliente

Caso o programa funcione em modo cliente, esta classe é responsável por enviar mensagens ao programa servidor que, na verdade, é o mesmo programa funcionando em outra máquina virtual, local ou remota que, está em modo servidor.

6.3.5. Classe Recebedor

Quando o programa Bichos do Tietê funciona em modo cliente, ele além de enviar mensagens ao servidor, deve também ter o atributo de receber eventuais mensagens enviadas pelo servidor, para atualização em tela e, esta classe tem a responsabilidade de fazer isso.

6.4. Pacote View



Este pacote é o responsável por mostrar a tela para o usuário do programa. Caso o programa Bichos do Tietê fosse criado para outro tipo de hardware, como um tablet ou smartphone, basicamente deveria ser desenvolvido novo pacote View;

6.4.1. Classe Timer Grava Catalogado

Quando o usuário registra um animal catalogado, o mesmo não é imediatamente gravado em disco. O motivo para isso é para economizar energia, pois o programa Bichos do Tietê funcionará em notebooks as margens

do rio Tietê, onde geralmente não há energia elétrica. Esta classe foi modelada para que, a cada 10 segundos, se faça uma verificação dos animais catalogados, serializem e armazenem em disco.

6.4.2. Classe Timer Quantidade Clientes Conectados

Quando o programa trabalha em modo servidor, exibe uma informação de quantas estações de trabalho estão conectadas na tela. Esta classe possui funções para verificar em tempos em tempos o número de usuários conectados e exibe em tela.

6.4.3. Classe Timer Verifica Conexao Cliente

Quando o programa opera em modo cliente, pode ser que a conexão se perca temporariamente com o servidor e, esta classe verifica esta conexão, o refaz se perdeu e exibe na tela a informação se está conectado ou não.

6.4.4. Classe View Catalogacao

Esta classe é responsável pela Interface do usuário do sistema. Nela estão definidos os botões de ações como catalogar animais e exportar os dados. Possui um campo que exibe os registros dos animais catalogados pela determinada estação e pelas outras também, independentes se o programa opera em modo cliente ou servidor.

7. Listagem do código fonte

```
package tipos;
import model.FuncoesGerais;
public enum Animal {
	capivara, mamiferosSilvestres, gato, cobra, avesSilvestres, anfibio, coruja, cachorro, nenhum;
	public static String getNomeImagemBig(Animal animal) {
		switch (animal) {
		case anfibio:
			return FuncoesGerais.getPathImagens() + "anfibio_big.png";
		case avesSilvestres:
			return FuncoesGerais.getPathImagens() + "avesilvestre_big.png";
		case cachorro:
			return FuncoesGerais.getPathImagens() + "cachorro_big.png";
```

```
                case capivara:
                        return FuncoesGerais.getPathImagens() + "capivara_big.png";
                case cobra:
                        return FuncoesGerais.getPathImagens() + "cobra_big.png";
                case coruja:
                        return FuncoesGerais.getPathImagens() + "coruja_big.png";
                case gato:
                        return FuncoesGerais.getPathImagens() + "gato_big.png";
                case mamiferosSilvestres:
                        return FuncoesGerais.getPathImagens() + "mamiferosilvestre_big.png";
                default:
                        return null;

                }
        }

        public static String getNomeImagemSmall(Animal animal) {
                switch (animal) {
                case anfibio:
                        return FuncoesGerais.getPathImagens() + "anfibio_small.png";
                case avesSilvestres:
                        return FuncoesGerais.getPathImagens() + "avesilvestre_small.png";
                case cachorro:
                        return FuncoesGerais.getPathImagens() + "cachorro_small.png";
                case capivara:
                        return FuncoesGerais.getPathImagens() + "capivara_small.png";
                case cobra:
                        return FuncoesGerais.getPathImagens() + "cobra_small.png";
                case coruja:
                        return FuncoesGerais.getPathImagens() + "coruja_small.png";
                case gato:
                        return FuncoesGerais.getPathImagens() + "gato_small.png";
                case mamiferosSilvestres:
                        return FuncoesGerais.getPathImagens() + "mamiferosilvestre_small.png";
                default:
                        return null;

                }
        }

        public static String getDescricao(Animal animal) {
                switch (animal) {
                case coruja:
                        return "Coruja";
                case capivara:
                        return "Capivara";
                case cobra:
                        return "Cobra";
                case anfibio:
                        return "Anfibio";
                case cachorro:
                        return "Cachorro";
                case gato:
                        return "Gato";
                case avesSilvestres:
                        return "Aves Silvestres";
                case mamiferosSilvestres:
                        return "Mamíferos Silvestres";
                default:
                        return null;

                }
        }
}
package tipos;
```

```
import java.io.Serializable;
import java.util.Date;

public class Catalogado implements Serializable {

	private static final long serialVersionUID = -6258428778056263021L;
	private Date dataRegistro;
	private Animal animal;
	private String observacao;
	private String localizacao;
	private String usuario;

	public String getUsuario() {
		return usuario;
	}
	public void setUsuario(String usuario) {
		this.usuario = usuario;
	}
	public Date getDataRegistro() {
		return dataRegistro;
	}
	public void setDataRegistro(Date date) {
		this.dataRegistro = date;
	}
	public Animal getAnimal() {
		return animal;
	}
	public void setAnimal(Animal animal) {
		this.animal = animal;
	}
	public String getObservacao() {
		return observacao;
	}
	public void setObservacao(String observacao) {
		this.observacao = observacao;
	}
	public String getLocalizacao() {
		return localizacao;
	}
	public void setLocalizacao(String localizacao) {
		this.localizacao = localizacao;
	}
	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((animal == null) ? 0 : animal.hashCode());
		result = prime * result
				+ ((dataRegistro == null) ? 0 : dataRegistro.hashCode());
		result = prime * result
				+ ((localizacao == null) ? 0 : localizacao.hashCode());
		result = prime * result
				+ ((observacao == null) ? 0 : observacao.hashCode());
		result = prime * result + ((usuario == null) ? 0 : usuario.hashCode());
		return result;
	}
	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Catalogado other = (Catalogado) obj;
		if (animal != other.animal)
```

```
                        return false;
                if (dataRegistro == null) {
                        if (other.dataRegistro != null)
                                return false;
                } else if (!dataRegistro.equals(other.dataRegistro))
                        return false;
                if (localizacao == null) {
                        if (other.localizacao != null)
                                return false;
                } else if (!localizacao.equals(other.localizacao))
                        return false;
                if (observacao == null) {
                        if (other.observacao != null)
                                return false;
                } else if (!observacao.equals(other.observacao))
                        return false;
                if (usuario == null) {
                        if (other.usuario != null)
                                return false;
                } else if (!usuario.equals(other.usuario))
                        return false;
                return true;
        }
        @Override
        public String toString() {
                return "Catalogado [dataRegistro=" + dataRegistro + ", animal="
                                + animal + ", observacao=" + observacao + ", local=" + localizacao
                                + "]";
        }


}
package controller;
import java.io.IOException;
import java.io.PrintStream;
import java.net.Socket;
import java.net.UnknownHostException;

import javax.swing.JTextArea;

public class Cliente {

        private String host;
        private int porta;
        private JTextArea textArea;
        private Socket socket;

        public boolean getEstahConectado() {
                return (socket != null && socket.isConnected());
        }

        public Cliente(String host, int porta, JTextArea textArea)
                        throws UnknownHostException, IOException {
                this.host = host;
                this.porta = porta;
                this.textArea = textArea;

                socket = new Socket(this.host, this.porta);

                Recebedor r = new Recebedor(socket.getInputStream(), this.textArea);
                new Thread(r).start();
        }

        public void enviarMensagem(String mensagem) throws UnknownHostException,
                        IOException {
```

```
                        // thread para receber mensagens do servidor
                        PrintStream saida = new PrintStream(socket.getOutputStream());
                        saida.println(mensagem);

                        textArea.setCaretPosition(textArea.getText().length());
                }
}
package tipos;

import java.io.Serializable;

public class Configuracao implements Serializable {
                        private static final long serialVersionUID = 20240751642709624871;
                private String localizacao;
                private String usuario;
                private String host; // endereço do servidor
                private String porta;
                private Boolean ehServidor;

                public Boolean getEhServidor() {
                        return ehServidor;
                }
                public void setEhServidor(Boolean ehServidor) {
                        this.ehServidor = ehServidor;
                }
                public String getPorta() {
                        return porta;
                }
                public void setPorta(String porta) {
                        this.porta = porta;
                }
                public String getHost() {
                        return host;
                }
                public void setHost(String host) {
                        this.host = host;
                }
                public String getLocalizacao() {
                        return localizacao;
                }
                public void setLocalizacao(String localizacao) {
                        this.localizacao = localizacao;
                }
                public String getUsuario() {
                        return usuario;
                }
                public void setUsuario(String usuario) {
                        this.usuario = usuario;
                }
                }
package controller;

import tipos.Animal;

public class ContaCatalogado {
                private int capivara = 0;
                private int mamiferosSilvestres = 0;
                private int gato = 0;
                private int cobra = 0;
                private int avesSilvestres = 0;
                private int anfibio = 0;
                private int coruja = 0;
                private int cachorro = 0;

                public void add(Animal animal) {
                        switch (animal) {
```

```
                        case anfibio:
                                this.anfibio++;
                                break;
                        case avesSilvestres:
                                this.avesSilvestres++;
                                break;
                        case cachorro:
                                this.cachorro++;
                                break;
                        case capivara:
                                this.capivara++;
                                break;
                        case cobra:
                                this.cobra++;
                                break;
                        case coruja:
                                this.coruja++;
                                break;
                        case gato:
                                this.gato++;
                                break;
                        case mamiferosSilvestres:
                                this.mamiferosSilvestres++;
                                break;
                        default:
                                break;
                        }
                }

                public int getCapivara() {
                        return capivara;
                }

                public int getMamiferosSilvestres() {
                        return mamiferosSilvestres;
                }

                public int getGato() {
                        return gato;
                }

                public int getCobra() {
                        return cobra;
                }

                public int getAvesSilvestres() {
                        return avesSilvestres;
                }

                public int getAnfibio() {
                        return anfibio;
                }

                public int getCoruja() {
                        return coruja;
                }

                public int getCachorro() {
                        return cachorro;
                }
}
package controller;

import java.io.IOException;
import java.net.UnknownHostException;
import java.text.SimpleDateFormat;
```

```
import java.util.ArrayList;
import java.util.GregorianCalendar;
import java.util.List;

import javax.swing.JOptionPane;
import javax.swing.JTextArea;
import javax.swing.text.BadLocationException;

import tipos.Animal;
import tipos.Catalogado;
import tipos.Configuracao;
import model.ExportaPlanilha;
import model.FuncoesGerais;
import model.GravaAnimalCatalogado;
import model.GravaConfiguracao;
import model.LerAnimalCatalogado;
import model.LerConfiguracao;

public class ControllerBichos {

	private List<Catalogado> listaCatalogado = new ArrayList<Catalogado>();
	private ContaCatalogado contaCatalogado = new ContaCatalogado();
	private JTextArea textArea;
	private Servidor servidor = null;
	private Cliente cliente = null;
	private boolean fecharSistema = false;

	SimpleDateFormat sdf = new SimpleDateFormat("E dd/MM/yyyy HH:mm:ss");

	public boolean getFecharSistema() {
		return fecharSistema;
	}

	public int getQuantidadeClientesConectados() {
		int auxiliar=0;

		if (servidor != null) {
			auxiliar = servidor.getQtdeClientes();
		}

		return auxiliar;
	}

	public void tratarServidor(Configuracao configuracao) throws IOException {

		if (!fecharSistema) {
			if (configuracao.getEhServidor()) {
				servidor = new Servidor(Integer.parseInt(configuracao
						.getPorta()), textArea);

				servidor.executa();
			}

			textArea.setCaretPosition(textArea.getText().length());

		}
	}

	public void conectarCliente(Configuracao configuracao) {
		if (!fecharSistema) {
			if (!configuracao.getEhServidor()
					&& !configuracao.getHost().equals("")
					&& !configuracao.getPorta().equals("")) {
				int porta = Integer.parseInt(configuracao.getPorta());
				try {
					cliente = new Cliente(configuracao.getHost(), porta, textArea);
```

```
                        } catch (UnknownHostException e) {
                                cliente = null;
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                cliente = null;
                        }
                }
        }
}

private String getLinhaCatalogado(Catalogado catalogado) {
        StringBuilder stb = new StringBuilder();
        stb.append(sdf.format(catalogado.getDataRegistro()) + " - ");
        stb.append(Animal.getDescricao(catalogado.getAnimal()) + " - ");
        stb.append("Local:" + catalogado.getLocalizacao() + " - ");
        stb.append("Usuário:" + catalogado.getUsuario());

        if (!catalogado.getObservacao().equals("")) {
                stb.append(" - Observação:" + catalogado.getObservacao());
        }

        return stb.toString();
}

private void addMemo(Catalogado catalogado) {
        textArea.append(getLinhaCatalogado(catalogado)+"\n");
        textArea.setCaretPosition(textArea.getText().length());

}

public ControllerBichos(JTextArea textArea) {
        this.textArea = textArea;

        this.fecharSistema = false;

        LerAnimalCatalogado arquivo = new LerAnimalCatalogado(
                        FuncoesGerais.getNomeArquivoCatalogado());
        try {
                arquivo.abrirArquivo();
                listaCatalogado = arquivo.getCatalogado();
                arquivo.fecharArquivo();

                for (Catalogado c : listaCatalogado) {
                        addMemo(c);
                }

                for (int i = 0; i < listaCatalogado.size(); i++) {
                        contaCatalogado.add(listaCatalogado.get(i).getAnimal());
                }
        } catch (Exception e) {
                listaCatalogado = new ArrayList<Catalogado>();
        }
}

public int getQtdeCatalogado(Animal animal) {
        switch (animal) {
        case anfibio:
                return contaCatalogado.getAnfibio();
        case avesSilvestres:
                return contaCatalogado.getAvesSilvestres();
        case cachorro:
                return contaCatalogado.getCachorro();
        case capivara:
                return contaCatalogado.getCapivara();
        case cobra:
                return contaCatalogado.getCobra();
```

```
                case coruja:
                        return contaCatalogado.getCoruja();
                case gato:
                        return contaCatalogado.getGato();
                case mamiferosSilvestres:
                        return contaCatalogado.getMamiferosSilvestres();
                default:
                        return -1;

                }
        }

        public void setConfiguracao(String localizacao, String usuario,
                        String host, String porta, Boolean ehServidor,
                        Configuracao configuracaoAnterior) throws Exception {
                Configuracao configuracao = new Configuracao();

                configuracao.setLocalizacao(localizacao);
                configuracao.setUsuario(usuario);
                configuracao.setHost(host);
                configuracao.setPorta(porta.toString());
                configuracao.setEhServidor(ehServidor);

                GravaConfiguracao gravaConfiguracao = new GravaConfiguracao();
                gravaConfiguracao.abrirArquivo();
                gravaConfiguracao.salvar(configuracao);
                gravaConfiguracao.fecharArquivo();

                if (!configuracaoAnterior.getPorta().equals(configuracao.getPorta())
                                || !configuracaoAnterior.getEhServidor().equals(
                                                configuracao.getEhServidor())) {
                        fecharSistema = true;

                        servidor = null;
                        cliente = null;

                        JOptionPane
                                        .showMessageDialog(
                                                        null,
                                                        "Troca de configuração de conexão. Favor fechar
e abrir o programa!",
                                                        "Informação",
JOptionPane.INFORMATION_MESSAGE);

                        System.exit(0);
                }
        }

        public Configuracao getConfiguracao() throws ClassNotFoundException,
                        IOException {
                LerConfiguracao lerConfiguracao = new LerConfiguracao();
                lerConfiguracao.abrirArquivo();
                Configuracao auxiliar = lerConfiguracao.getConfiguracao();
                lerConfiguracao.fecharArquivo();

                return auxiliar;
        }

        public void catalogar(Animal animal, Configuracao configuracao,
                        String observacao) throws IOException, BadLocationException {
                Catalogado catalogado = new Catalogado();
                catalogado.setObservacao(observacao);
                catalogado.setAnimal(animal);
                catalogado.setLocalizacao(configuracao.getLocalizacao());
                catalogado.setUsuario(configuracao.getUsuario());
                catalogado.setDataRegistro(new GregorianCalendar().getTime());
```

```
                listaCatalogado.add(catalogado);

                contaCatalogado.add(catalogado.getAnimal());

                addMemo(catalogado);

                if (configuracao.getEhServidor()) {
                        servidor.distribuiMensagem(getLinhaCatalogado(catalogado));
                } else {
                        if (cliente != null) {
                                if (!cliente.getEstahConectado()) {
                                        int porta = Integer.parseInt(configuracao.getPorta());
                                        cliente = new Cliente(configuracao.getHost(), porta,
                                                        textArea);
                                }
                        }

                        if (cliente != null && cliente.getEstahConectado()) {
                                cliente.enviarMensagem(getLinhaCatalogado(catalogado)+"\n");
                        }
                }
        }

        public void setListaCatalogado() throws IOException {
                GravaAnimalCatalogado gravaAnimalCatalogado = new GravaAnimalCatalogado(
                                FuncoesGerais.getNomeArquivoCatalogado());
                gravaAnimalCatalogado.abrirArquivo();

                gravaAnimalCatalogado.setLista(listaCatalogado);

                gravaAnimalCatalogado.fecharArquivo();

        }

        public String getPathIconeExcel() {
                return FuncoesGerais.getPathImagens() + "excel.png";
        }

        public void exportarPlanilha(String arquivo) throws IOException {
                new ExportaPlanilha(listaCatalogado, arquivo);
        }

        public boolean getClienteEstahConectado() {
                return (cliente != null && cliente.getEstahConectado());
        }
}
package model;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;

import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import tipos.Animal;
import tipos.Catalogado;
```

```
public class ExportaPlanilha {

	SimpleDateFormat sdfDiaSemana = new SimpleDateFormat("E");
	SimpleDateFormat sdfData = new SimpleDateFormat("dd/MM/yyyy");
	SimpleDateFormat sdfHora = new SimpleDateFormat("HH:mm");

	XSSFWorkbook workbook = new XSSFWorkbook();
	XSSFSheet sheet = workbook.createSheet("Animais catalogados");

	public ExportaPlanilha(List<Catalogado> listaCatalogado, String nomePlanilha) throws IOException {
		super();

		Map<String, Object[]> data = new TreeMap<String, Object[]>();

		data.put("1", new Object[] { "Posição", "Dia Semana", "Data", "Hora",
				"Local", "Animal", "Usuário", "Observação" });

		Integer cont = 2;

		for (int i = 0; i < listaCatalogado.size(); i++) {
			data.put(
					cont.toString(),
					new Object[] {
							i + 1,
							sdfDiaSemana.format(listaCatalogado.get(i)
									.getDataRegistro()),
							sdfData.format(listaCatalogado.get(i)
									.getDataRegistro()),
							sdfHora.format(listaCatalogado.get(i)
									.getDataRegistro()),
							listaCatalogado.get(i).getLocalizacao(),
							Animal.getDescricao(listaCatalogado.get(i)
									.getAnimal()),
							listaCatalogado.get(i).getUsuario(),
							listaCatalogado.get(i).getObservacao() });

			cont++;
		}

		// Iterate over data and write to sheet
		Set<String> keyset = data.keySet();
		int rownum = 0;
		for (String key : keyset) {
			Row row = sheet.createRow(rownum++);
			Object[] objArr = data.get(key);
			int cellnum = 0;
			for (Object obj : objArr) {
				Cell cell = row.createCell(cellnum++);
				if (obj instanceof String)
					cell.setCellValue((String) obj);
				else if (obj instanceof Integer)
					cell.setCellValue((Integer) obj);
			}
		}

		FileOutputStream out = new FileOutputStream(new File(nomePlanilha));
		workbook.write(out);
		out.close();
	}

}
package model;

import javax.swing.JTextArea;
import javax.swing.text.BadLocationException;
```

```
public class FuncoesGerais {

	private static String pathSistema = System.getProperty("user.dir")+"\\";

	private static String pathBaseDados = pathSistema + "basedados\\";
	private static String pathImagens = pathSistema + "imagens\\";

	public static String getPathSistema() {
		return pathSistema;
	}

	public static String getNomeArquivoCatalogado (){
		return pathBaseDados + "animais.dat";
	}

	public static String getPathImagens() {
		return pathImagens;
	}

	public static boolean podeIncluirLinhaTextArea(JTextArea textArea, String msg) throws
BadLocationException {
		int qtdeLinhas = textArea.getLineCount();
		boolean encontrou = false;

		for(int i = 0; i < qtdeLinhas; i++) {
			int inicio = textArea.getLineStartOffset(i);
			int fim = textArea.getLineEndOffset(i);

			String linha = textArea.getText(inicio, fim - inicio);

			if (linha.equals(msg)) {
				encontrou = true;
				break;
			}
		}

		return !encontrou;
	}

}
package model;

import java.io.IOException;
import java.util.List;

import tipos.Catalogado;

public class GravaAnimalCatalogado extends GravaArquivo {

	public GravaAnimalCatalogado(String nomeArquivo) {
		super(nomeArquivo);
	}

	public void setLista(List<Catalogado> lista) throws IOException {
		for(int i=0; i < lista.size(); i++) {
			output.writeObject(lista.get(i));
		}

	}
}
package model;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
```

```
public class GravaArquivo {
	protected ObjectOutputStream output;
	private String nomeArquivo;

	public GravaArquivo(String nomeArquivo) {
		super();
		this.nomeArquivo = nomeArquivo;
	}

	public void abrirArquivo() throws IOException {
		FileOutputStream file = new FileOutputStream(nomeArquivo);
		output = new ObjectOutputStream(file);
	}

	public void fecharArquivo() throws IOException {
		if (output != null) {
			output.close();
		}
	}


}
package model;

import tipos.Configuracao;

public class GravaConfiguracao extends GravaArquivo {

	public GravaConfiguracao() {
		super(FuncoesGerais.getPathSistema() + "config.dat");
	}

	public void salvar(Configuracao configuracao) throws Exception {

		configuracao.setLocalizacao(configuracao.getLocalizacao().trim());
		configuracao.setUsuario(configuracao.getUsuario().trim());

		if (configuracao.getLocalizacao().isEmpty()) {
			throw new Exception("Localização não pode ser vazia!");
		}

		if (configuracao.getUsuario().isEmpty()) {
			throw new Exception("Usuário não pode ser vazio!");
		}

		output.writeObject(configuracao);
	}
}
package model;

import java.io.EOFException;
import java.util.ArrayList;
import java.util.List;

import tipos.Catalogado;

public class LerAnimalCatalogado extends LerArquivo {

	public LerAnimalCatalogado(String nomeArquivo) {
		super(nomeArquivo);
	}

	public List<Catalogado> getCatalogado() throws Exception {
		List<Catalogado> lista = new ArrayList<Catalogado>();

		Catalogado registro;
```

```
                try {
                        while (true) {
                                registro = (Catalogado) input.readObject();

                                lista.add(registro);
                        }
                } catch (EOFException endOfFileException) {
                        return lista;
                } catch (Exception exception) {
                        throw new Exception(exception.getMessage());
                }

        }

}
package model;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class LerArquivo {

        protected ObjectInputStream input;
        private String nomeArquivo;

        public void abrirArquivo() throws IOException {
                FileInputStream file = new FileInputStream(nomeArquivo);
                input = new ObjectInputStream(file);
        }

        public LerArquivo(String nomeArquivo) {
                super();
                this.nomeArquivo = nomeArquivo;
        }

        public void fecharArquivo() throws IOException {
                if (input != null) {
                        input.close();
                }
        }

}
package model;

import java.io.IOException;

import tipos.Configuracao;

public class LerConfiguracao extends LerArquivo {

        public LerConfiguracao() {
                super(FuncoesGerais.getPathSistema() + "config.dat");
        }

        public Configuracao getConfiguracao() throws ClassNotFoundException, IOException {
                Configuracao registro = (Configuracao) input.readObject();

                return registro;
        }

}
package controller;

import java.io.InputStream;
```

```
import java.util.Scanner;

import javax.swing.JTextArea;
import javax.swing.text.BadLocationException;

import model.FuncoesGerais;

public class Recebedor implements Runnable {

	private InputStream servidor;
	private JTextArea textArea;

	public Recebedor(InputStream servidor, JTextArea textArea) {
		this.servidor = servidor;
		this.textArea = textArea;
	}

	public void run() {
		// recebe msgs do servidor e imprime na tela
		Scanner s = new Scanner(this.servidor);
		while (s.hasNextLine()) {
			String msg = s.nextLine()+"\n";

			try {
				if (FuncoesGerais.podeIncluirLinhaTextArea(textArea, msg)) {
					textArea.append(msg);
				}
			} catch (BadLocationException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}

		textArea.setCaretPosition(textArea.getText().length());
	}
}package controller;

import java.io.IOException;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;

import javax.swing.JTextArea;
import javax.swing.text.BadLocationException;

import model.FuncoesGerais;

public class Servidor {

	private int porta;
	private List<PrintStream> clientes;
	private JTextArea textArea;

	public Servidor(int porta, JTextArea textArea) {
		this.porta = porta;
		this.textArea = textArea;
		this.clientes = new ArrayList<PrintStream>();
	}

	public void executa() throws IOException {
		ServerSocket servidor = new ServerSocket(this.porta);
		textArea.append("Conexão aberta através da porta " + porta + ".\n");

		textArea.setCaretPosition(textArea.getText().length());
```

```
                while (true) {
                        Socket cliente = servidor.accept();
                        textArea.append("Nova conexão com o cliente "
                                        + cliente.getInetAddress().getHostAddress()+"\n");

                        PrintStream ps = new PrintStream(cliente.getOutputStream());
                        this.clientes.add(ps);

                        TrataCliente tc = new TrataCliente(cliente.getInputStream(), this);
                        new Thread(tc).start();

                        textArea.setCaretPosition(textArea.getText().length());
                }
        }

        public void distribuiMensagem(String msg) throws BadLocationException {
                // envia msg para todo mundo
                if (FuncoesGerais.podeIncluirLinhaTextArea(textArea, msg+"\n")) {
                        textArea.append(msg+"\n");
                }

                for (PrintStream p : this.clientes) {
                        p.println(msg+"\n");
                }

                textArea.setCaretPosition(textArea.getText().length());
        }

        public int getQtdeClientes () {
                return clientes.size();
        }
}package view;

import java.awt.event.ActionListener;
import java.io.IOException;

import javax.swing.Timer;

import controller.ControllerBichos;

public class TimerGravaCatalogado {

        private Timer timer;

        private ControllerBichos controller;

        public TimerGravaCatalogado(ControllerBichos controller) {
                super();

                ActionListener action = new ActionListener() {
                        public void actionPerformed(java.awt.event.ActionEvent e) {

                                try {
                                        executar();
                                } catch (IOException e1) {
                                        e1.printStackTrace();
                                }
                        }
                };

                this.controller = controller;

                this.timer = new Timer(1000*10, action); // rodar a cada 10 segundos
                timer.start();
        }
```

```
		private void executar() throws IOException {
			controller.setListaCatalogado();
		}

}
package view;

import java.awt.Color;
import java.awt.event.ActionListener;

import javax.swing.JLabel;
import javax.swing.Timer;

import controller.ControllerBichos;

public class TimerQuantidadeClientesConectados {

		private ControllerBichos controller;
		private JLabel lblEstadoConexao;
		private Timer timer;

		private void executar() {
			lblEstadoConexao.setForeground(Color.BLUE);

			int auxiliar = controller.getQuantidadeClientesConectados();

			lblEstadoConexao.setText("CLIENTES CONECTADOS: " + auxiliar);
		}

		public TimerQuantidadeClientesConectados(ControllerBichos controller,
				JLabel lblEstadoConexao) {
			this.controller = controller;
			this.lblEstadoConexao = lblEstadoConexao;

			ActionListener action = new ActionListener() {
				public void actionPerformed(java.awt.event.ActionEvent e) {

					executar();
				}
			};

			this.timer = new Timer(1000 * 2, action); // rodar a cada 2 segundos
			timer.start();
		}

}
package view;

import java.awt.Color;
import java.awt.event.ActionListener;
import java.io.IOException;

import javax.swing.JLabel;
import javax.swing.Timer;

import controller.ControllerBichos;

public class TimerVerificaConexaoCliente {

		private ControllerBichos controller;
		private JLabel lblEstadoConexao;
		private Timer timer;

		private void executar() {
			if (controller.getClienteEstahConectado()) {
```

```
                        lblEstadoConexao.setForeground(Color.GREEN);
                        lblEstadoConexao.setText("CONECTADO COM O SERVIDOR");
                } else {
                        lblEstadoConexao.setForeground(Color.red);
                        lblEstadoConexao.setText("SEM CONEXÃO COM O SERVIDOR");
                }
        }

        public TimerVerificaConexaoCliente(ControllerBichos controller,
                        JLabel lblEstadoConexao) {
                this.controller = controller;
                this.lblEstadoConexao = lblEstadoConexao;

                ActionListener action = new ActionListener() {
                        public void actionPerformed(java.awt.event.ActionEvent e) {

                                executar();
                        }
                };

                this.timer = new Timer(1000*2, action); // rodar a cada 2 segundos
                timer.start();
        }
}
package controller;
import java.io.InputStream;
import java.util.Scanner;

import javax.swing.text.BadLocationException;

public class TrataCliente implements Runnable {

  private InputStream cliente;
  private Servidor servidor;

  public TrataCliente(InputStream cliente, Servidor servidor) {
    this.cliente = cliente;
    this.servidor = servidor;
  }

  public void run() {
    // quando chegar uma msg, distribui pra todos
    Scanner s = new Scanner(this.cliente);
    while (s.hasNextLine()) {
      try {
                servidor.distribuiMensagem(s.nextLine());
        } catch (BadLocationException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
    }
    s.close();
  }
 }package view;

import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
```

```
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.filechooser.FileFilter;

import controller.ControllerBichos;

import tipos.Animal;
import tipos.Configuracao;

import java.io.File;
import java.io.IOException;
import java.text.ParseException;

public class ViewCatalogacao implements ActionListener {

	private ControllerBichos controller;
	private JFrame frmJanela;

	private Configuracao configuracao;

	private JButton btnAnfibio;
	private JButton btnAvesSilvestres;
	private JButton btnCachorro;
	private JButton btnCapivara;
	private JButton btnCobra;
	private JButton btnCoruja;
	private JButton btnGato;
	private JButton btnMamiferosSilvestres;

	private JLabel lblCatalogadoAnfibio;
	private JLabel lblCatalogadoAvesSilvestres;
	private JLabel lblCatalogadoCachorro;
	private JLabel lblCatalogadoCapivara;
	private JLabel lblCatalogadoCobra;
	private JLabel lblCatalogadoCoruja;
	private JLabel lblCatalogadoGato;
	private JLabel lblCatalogadoMamiferosSilvestres;

	private JButton btnExportarExcel;
	private JLabel lblEstadoConexao;

	private JLabel lblLocalizacao;
	private JTextField edtLocalizacao;

	private JLabel lblUsuario;
	private JTextField edtUsuario;

	private JLabel lblHost;
	private JTextField edtHost;

	private JLabel lblPorta;
	private JTextField edtPorta;

	private JCheckBox ckbEhServidor;

	private JTextArea textArea;
	private JScrollPane jscroolPane;

	public ViewCatalogacao() throws ParseException {
	}

	private void carregarConfiguracao() throws ClassNotFoundException,
				IOException {
		configuracao = controller.getConfiguracao();
```

```
		edtLocalizacao.setText(configuracao.getLocalizacao());
		edtUsuario.setText(configuracao.getUsuario());
		edtHost.setText(configuracao.getHost());
		edtPorta.setText(configuracao.getPorta());
		ckbEhServidor.setSelected(configuracao.getEhServidor());
	}

	public static void main(String[] args) throws ClassNotFoundException,
			IOException {
		ViewCatalogacao b;
		try {
			b = new ViewCatalogacao();
			b.janela();
		} catch (ParseException e) {
			e.printStackTrace();
		}
	}

	public void janela() throws ClassNotFoundException, IOException {
		frmJanela = new JFrame("Projeto Bichos do Tietê - 2014");
		frmJanela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frmJanela.setResizable(false);
		frmJanela.setLayout(null);
		frmJanela.setBackground(Color.darkGray);

		textArea = new JTextArea(5, 200);
		textArea.setColumns(60);
		textArea.setLineWrap(true);
		textArea.setRows(18);
		textArea.setEditable(false);
		textArea.setWrapStyleWord(true);
		textArea.setForeground(Color.green);
		textArea.setBackground(Color.black);

		jscroolPane = new JScrollPane(textArea);

		JPanel jpnContainer = new JPanel();
		jpnContainer.setBounds(0, 0, 700, 630);
		jpnContainer.setLayout(null);
		jpnContainer.setBackground(Color.darkGray);

		JPanel areaConfig = new JPanel();
		areaConfig.setBounds(05, 05, 685, 50);
		areaConfig.setBackground(Color.BLACK);
		areaConfig.setLayout(null);

		JPanel areaBotoes = new JPanel();
		areaBotoes.setBounds(05, 59, 685, 115);
		areaBotoes.setBackground(Color.BLACK);
		areaBotoes.setLayout(null);

		JPanel areaMemo = new JPanel();
		areaMemo.setBounds(05, 177, 685, 300);
		areaMemo.setBackground(Color.BLACK);

		JPanel areaOperacao = new JPanel();
		areaOperacao.setBounds(05, 475, 685, 470);
		areaOperacao.setBackground(Color.BLACK);
		areaOperacao.setLayout(null);

		int left;
		int width;

		left = 3;
		lblLocalizacao = new JLabel("Localização:");
		lblLocalizacao.setLocation(left, 10);
```

```
lblLocalizacao.setSize(75, 15);
lblLocalizacao.setForeground(Color.white);

lblHost = new JLabel("Host:");
lblHost.setLocation(left + 42, 30);
lblHost.setSize(75, 15);
lblHost.setForeground(Color.white);

left += 75;
edtLocalizacao = new JTextField();
edtLocalizacao.setLocation(left, 10);
edtLocalizacao.setSize(220, 15);
edtLocalizacao.setForeground(Color.white);
edtLocalizacao.setBackground(Color.black);

edtHost = new JTextField();
edtHost.setLocation(left, 30);
edtHost.setSize(120, 15);
edtHost.setForeground(Color.white);
edtHost.setBackground(Color.black);

left += 240;
lblUsuario = new JLabel("Usuário:");
lblUsuario.setLocation(left, 10);
lblUsuario.setSize(50, 15);
lblUsuario.setForeground(Color.white);

lblPorta = new JLabel("Porta:");
lblPorta.setLocation(left + 13, 30);
lblPorta.setSize(50, 15);
lblPorta.setForeground(Color.white);

left += 50;
edtUsuario = new JTextField();
edtUsuario.setLocation(left, 10);
edtUsuario.setSize(220, 15);
edtUsuario.setForeground(Color.white);
edtUsuario.setBackground(Color.black);

edtPorta = new JTextField();
edtPorta.setLocation(left, 30);
edtPorta.setSize(40, 15);
edtPorta.setForeground(Color.white);
edtPorta.setBackground(Color.black);

ckbEhServidor = new JCheckBox("Esta máquina é o Servidor", false);
ckbEhServidor.setLocation(left + 50, 30);
ckbEhServidor.setForeground(Color.white);
ckbEhServidor.setBackground(Color.black);
ckbEhServidor.setSize(190, 15);
ckbEhServidor.addActionListener(this);

left = 3;
width = 83;

btnAnfibio = new JButton("");
btnAnfibio.setIcon(new javax.swing.ImageIcon(Animal
                    .getNomeImagemSmall(Animal.anfibio)));
btnAnfibio.setBounds(left, 10, width, width);
btnAnfibio.setToolTipText("Catalogar um Anfíbio");
btnAnfibio.addActionListener(this);

lblCatalogadoAnfibio = new JLabel("Qtde:");
lblCatalogadoAnfibio.setLocation(left, 95);
lblCatalogadoAnfibio.setSize(width, 15);
lblCatalogadoAnfibio.setForeground(Color.white);
```

```
left += width + 2;
btnAvesSilvestres = new JButton("");
btnAvesSilvestres.setIcon(new javax.swing.ImageIcon(Animal
                .getNomeImagemSmall(Animal.avesSilvestres)));
btnAvesSilvestres.setBounds(left, 10, width, width);
btnAvesSilvestres.setToolTipText("Catalogar uma Ave Silvestre");
btnAvesSilvestres.addActionListener(this);

lblCatalogadoAvesSilvestres = new JLabel("Qtde:");
lblCatalogadoAvesSilvestres.setLocation(left, 95);
lblCatalogadoAvesSilvestres.setSize(width, 15);
lblCatalogadoAvesSilvestres.setForeground(Color.white);

left += width + 2;
btnCachorro = new JButton("");
btnCachorro.setIcon(new javax.swing.ImageIcon(Animal
                .getNomeImagemSmall(Animal.cachorro)));
btnCachorro.setBounds(left, 10, width, width);
btnCachorro.setToolTipText("Catalogar um Cachorro");
btnCachorro.addActionListener(this);

lblCatalogadoCachorro = new JLabel("Qtde:");
lblCatalogadoCachorro.setLocation(left, 95);
lblCatalogadoCachorro.setSize(width, 15);
lblCatalogadoCachorro.setForeground(Color.white);

left += width + 2;
btnCapivara = new JButton("");
btnCapivara.setIcon(new javax.swing.ImageIcon(Animal
                .getNomeImagemSmall(Animal.capivara)));
btnCapivara.setBounds(left, 10, width, width);
btnCapivara.setToolTipText("Catalogar uma Capivara");
btnCapivara.addActionListener(this);

lblCatalogadoCapivara = new JLabel("Qtde:");
lblCatalogadoCapivara.setLocation(left, 95);
lblCatalogadoCapivara.setSize(width, 15);
lblCatalogadoCapivara.setForeground(Color.white);

left += width + 2;
btnCobra = new JButton("");
btnCobra.setIcon(new javax.swing.ImageIcon(Animal
                .getNomeImagemSmall(Animal.cobra)));
btnCobra.setBounds(left, 10, width, width);
btnCobra.setToolTipText("Catalogar uma Cobra");
btnCobra.addActionListener(this);

lblCatalogadoCobra = new JLabel("Qtde:");
lblCatalogadoCobra.setLocation(left, 95);
lblCatalogadoCobra.setSize(width, 15);
lblCatalogadoCobra.setForeground(Color.white);

left += width + 2;
btnCoruja = new JButton("");
btnCoruja.setIcon(new javax.swing.ImageIcon(Animal
                .getNomeImagemSmall(Animal.coruja)));
btnCoruja.setBounds(left, 10, width, width);
btnCoruja.setToolTipText("Catalogar uma Coruja");
btnCoruja.addActionListener(this);

lblCatalogadoCoruja = new JLabel("Qtde:");
lblCatalogadoCoruja.setLocation(left, 95);
lblCatalogadoCoruja.setSize(width, 15);
lblCatalogadoCoruja.setForeground(Color.white);
```

```
left += width + 2;
btnGato = new JButton("");
btnGato.setIcon(new javax.swing.ImageIcon(Animal
                .getNomeImagemSmall(Animal.gato)));
btnGato.setBounds(left, 10, width, width);
btnGato.setToolTipText("Catalogar um Gato");
btnGato.addActionListener(this);

lblCatalogadoGato = new JLabel("Qtde:");
lblCatalogadoGato.setLocation(left, 95);
lblCatalogadoGato.setSize(width, 15);
lblCatalogadoGato.setForeground(Color.white);

left += width + 2;
btnMamiferosSilvestres = new JButton("");
btnMamiferosSilvestres.setIcon(new javax.swing.ImageIcon(Animal
                .getNomeImagemSmall(Animal.mamiferosSilvestres)));
btnMamiferosSilvestres.setBounds(left, 10, width, width);
btnMamiferosSilvestres
                .setToolTipText("Catalogar um Mamífero Silvestre");
btnMamiferosSilvestres.addActionListener(this);

lblCatalogadoMamiferosSilvestres = new JLabel("Qtde:");
lblCatalogadoMamiferosSilvestres.setLocation(left, 95);
lblCatalogadoMamiferosSilvestres.setSize(width, 15);
lblCatalogadoMamiferosSilvestres.setForeground(Color.white);

controller = new ControllerBichos(textArea);
new TimerGravaCatalogado(controller);

left = 3;

btnExportarExcel = new JButton("");
btnExportarExcel.setIcon(new javax.swing.ImageIcon(controller
                .getPathIconeExcel()));
btnExportarExcel.setBounds(left, 10, width, width);
btnExportarExcel
                .setToolTipText("Exportar animais catalogados para uma planilha Excel");
btnExportarExcel.addActionListener(this);

lblEstadoConexao = new JLabel("");
lblEstadoConexao.setLocation(left+150, 10);
lblEstadoConexao.setSize(300, 15);

// == ADICIONAR COMPONENTES =================
frmJanela.getContentPane().add(areaConfig);
frmJanela.getContentPane().add(areaBotoes);
frmJanela.getContentPane().add(areaMemo);
frmJanela.getContentPane().add(areaOperacao);

areaBotoes.add(btnAnfibio);
areaBotoes.add(lblCatalogadoAnfibio);
areaBotoes.add(btnAvesSilvestres);
areaBotoes.add(lblCatalogadoAvesSilvestres);
areaBotoes.add(btnCachorro);
areaBotoes.add(lblCatalogadoCachorro);
areaBotoes.add(btnCapivara);
areaBotoes.add(lblCatalogadoCapivara);
areaBotoes.add(btnCobra);
areaBotoes.add(lblCatalogadoCobra);
areaBotoes.add(btnCoruja);
areaBotoes.add(lblCatalogadoCoruja);
areaBotoes.add(btnGato);
areaBotoes.add(lblCatalogadoGato);
areaBotoes.add(btnMamiferosSilvestres);
areaBotoes.add(lblCatalogadoMamiferosSilvestres);
```

```
                areaConfig.add(lblLocalizacao);
                areaConfig.add(edtLocalizacao);
                areaConfig.add(lblUsuario);
                areaConfig.add(edtUsuario);
                areaConfig.add(lblHost);
                areaConfig.add(edtHost);
                areaConfig.add(lblPorta);
                areaConfig.add(edtPorta);
                areaConfig.add(ckbEhServidor);

                areaMemo.add(jscroolPane);

                areaOperacao.add(btnExportarExcel);
                areaOperacao.add(lblEstadoConexao);

                frmJanela.getContentPane().add(jpnContainer);
                // ====================

                frmJanela.setSize(700, 600);
                frmJanela.setLocationRelativeTo(null);
                frmJanela.setVisible(true);

                carregarConfiguracao();
                stualizarLabelQuantidadeRegistrada();

                textArea.setCaretPosition(textArea.getText().length());

                if (configuracao.getEhServidor()) {
                        new TimerQuantidadeClientesConectados(controller, lblEstadoConexao);
                        controller.tratarServidor(configuracao);
                }
                else {
                        controller.conectarCliente(configuracao);
                        new TimerVerificaConexaoCliente(controller, lblEstadoConexao);
                }
        }

        @Override
        public void actionPerformed(ActionEvent e) {
                try {

                        Animal animal = Animal.nenhum;

                        if (e.getSource() == ckbEhServidor) {
                                controller.setConfiguracao(edtLocalizacao.getText(),
                                                edtUsuario.getText(), edtHost.getText(),
                                                edtPorta.getText(), ckbEhServidor.isSelected(),
configuracao);
                                carregarConfiguracao();
                        }
                        else if (e.getSource() == btnExportarExcel) {
                                exportarExcel();
                        } else if (e.getSource() == btnAnfibio) {
                                animal = Animal.anfibio;
                        } else if (e.getSource() == btnAvesSilvestres) {
                                animal = Animal.avesSilvestres;
                        } else if (e.getSource() == btnCachorro) {
                                animal = Animal.cachorro;
                        } else if (e.getSource() == btnCapivara) {
                                animal = Animal.capivara;
                        } else if (e.getSource() == btnCobra) {
                                animal = Animal.cobra;
                        } else if (e.getSource() == btnCoruja) {
                                animal = Animal.coruja;
                        } else if (e.getSource() == btnGato) {
```

```
                        animal = Animal.gato;
                    } else if (e.getSource() == btnMamiferosSilvestres) {
                        animal = Animal.mamiferosSilvestres;
                    }

                    if (animal != Animal.nenhum) {

                        if (!controller.getFecharSistema()) {
                            controller.setConfiguracao(edtLocalizacao.getText(),
                                    edtUsuario.getText(), edtHost.getText(),
                                    edtPorta.getText(), ckbEhServidor.isSelected(),
configuracao);
                            carregarConfiguracao();

                            String observacao = JOptionPane
                                    .showInputDialog("Observação sobre "
                                            +
Animal.getDescricao(animal));

                            if (observacao != null) {
                                observacao = observacao.trim();
                                controller.catalogar(animal, configuracao, observacao);

                                stualizarLabelQuantidadeRegistrada();
                            }


                        }

                    }

                } catch (Exception e1) {
                    JOptionPane
                            .showMessageDialog(frmJanela, e1.getMessage(), "Erro", 0);
                }
            }

            private void stualizarLabelQuantidadeRegistrada() {
                lblCatalogadoAnfibio.setText("Qtd: "
                        + controller.getQtdeCatalogado(Animal.anfibio));
                lblCatalogadoAvesSilvestres.setText("Qtd: "
                        + controller.getQtdeCatalogado(Animal.avesSilvestres));
                lblCatalogadoCachorro.setText("Qtd: "
                        + controller.getQtdeCatalogado(Animal.cachorro));
                lblCatalogadoCapivara.setText("Qtd: "
                        + controller.getQtdeCatalogado(Animal.capivara));
                lblCatalogadoCobra.setText("Qtd: "
                        + controller.getQtdeCatalogado(Animal.cobra));
                lblCatalogadoCoruja.setText("Qtd: "
                        + controller.getQtdeCatalogado(Animal.coruja));
                lblCatalogadoGato.setText("Qtd: "
                        + controller.getQtdeCatalogado(Animal.gato));
                lblCatalogadoMamiferosSilvestres.setText("Qtd: "
                        + controller.getQtdeCatalogado(Animal.mamiferosSilvestres));
            }

            private void exportarExcel() {
                try {
                    JFileChooser saveFile = new JFileChooser();// new save dialog
                    saveFile.setAcceptAllFileFilterUsed(false);
                    saveFile.setDialogType(JFileChooser.SAVE_DIALOG);
                    saveFile.setDialogTitle("Exportar animais catalogados");

                    saveFile.addChoosableFileFilter(new FileFilter() {

                        String description = "Excel (*.xlsx)";
```

```
                        String extension = "xlsx";

                        public String getDescription() {
                                return description;
                        }

                        public boolean accept(File f) {
                                if (f == null)
                                        return false;
                                if (f.isDirectory())
                                        return true;
                                return f.getName().toLowerCase().endsWith(extension);
                        }
                });
                saveFile.setCurrentDirectory(new File("*.srt"));
                int result = saveFile.showSaveDialog(frmJanela);

                if (result == JFileChooser.APPROVE_OPTION) {
                        String strFileName = saveFile.getSelectedFile().getName();

                        if (!strFileName.isEmpty()) {
                                if (!strFileName.toLowerCase().endsWith("xlsx")) {
                                        throw new IOException(
                                                        "Arquivo deve ter a extensão xlsx");
                                }

                                String nomeArquivo = saveFile.getSelectedFile()
                                                .getAbsoluteFile().toString();

                                controller.exportarPlanilha(nomeArquivo);

                                JOptionPane.showMessageDialog(frmJanela, "Arquivo "
                                                + nomeArquivo + " criado!", "Informação",
                                                JOptionPane.INFORMATION_MESSAGE);
                        }
                }

        } catch (Exception er) {
                JOptionPane
                                .showMessageDialog(frmJanela, er.getMessage(), "Erro", 0);
        }

    }

}
```

8. Referencias Bibliográficas

8.1 LIVROS

Rede de Computadores, Quarta Edição, Andrew S. Tanenbaum, Editora Campus, Ano 2003, ISBN 8535211853.

Bluetooth Application Programming with the Java APIs Essentials Edition, Timothy J. Thompson, Editora Morgan Kaufmann Publishers, Ano 2004, ISBN 0123743427.

8.2 Acessos de links via Internet

<http://www.suapesquisa.com/pesquisa/riotiete.htm> Acesso em 22/04/2014.

<http://www.daee.sp.gov.br/index.php?option=com_content&view=article&id=793:historico-do-rio-tiete&catid=48:noticias&Itemid=53> Acesso em 22/04/2014.

<http://www.salesopolis.sp.gov.br/site2/> Acesso em 23/04/2014.

<http://site.sabesp.com.br/site/interna/default.aspx?secaoId=81> Acesso em 23/04/2014.

<http://www.daee.sp.gov.br/index.php?option=com_content&view=article&id=793:historico-do-rio-tiete&catid=48:noticias&Itemid=53> Acesso em 25/04/2014.

<http://www.goiania.go.gov.br/sistemas/scmag/dados/RefAutor/RefAutor21.pdf > Acesso em 27/04/2014.

<http://unipvirtual.com.br/material/RECUPERACAO/EAD/FUNDAMENTOS_REDES_DADOS_COMUNICACAO/PDF/geral_pdf.pdf > Acesso em 02/05/2014.

<http://www.caelum.com.br/apostila-java-orientacao-objetos/apendice-sockets/> Acesso em 03/05/2014.

<http://pt.wikipedia.org/wiki/Soquete_de_rede> Acesso em 09/05/2014.

<http://www.guj.com.br/> Acesso em 10/05/2014.