Trabalho Final de Engenharia de Componentes de Software

Alunos: Gian Ricardo Berkenbrock
& Eduardo Dockhorn da Costa

Roteiro

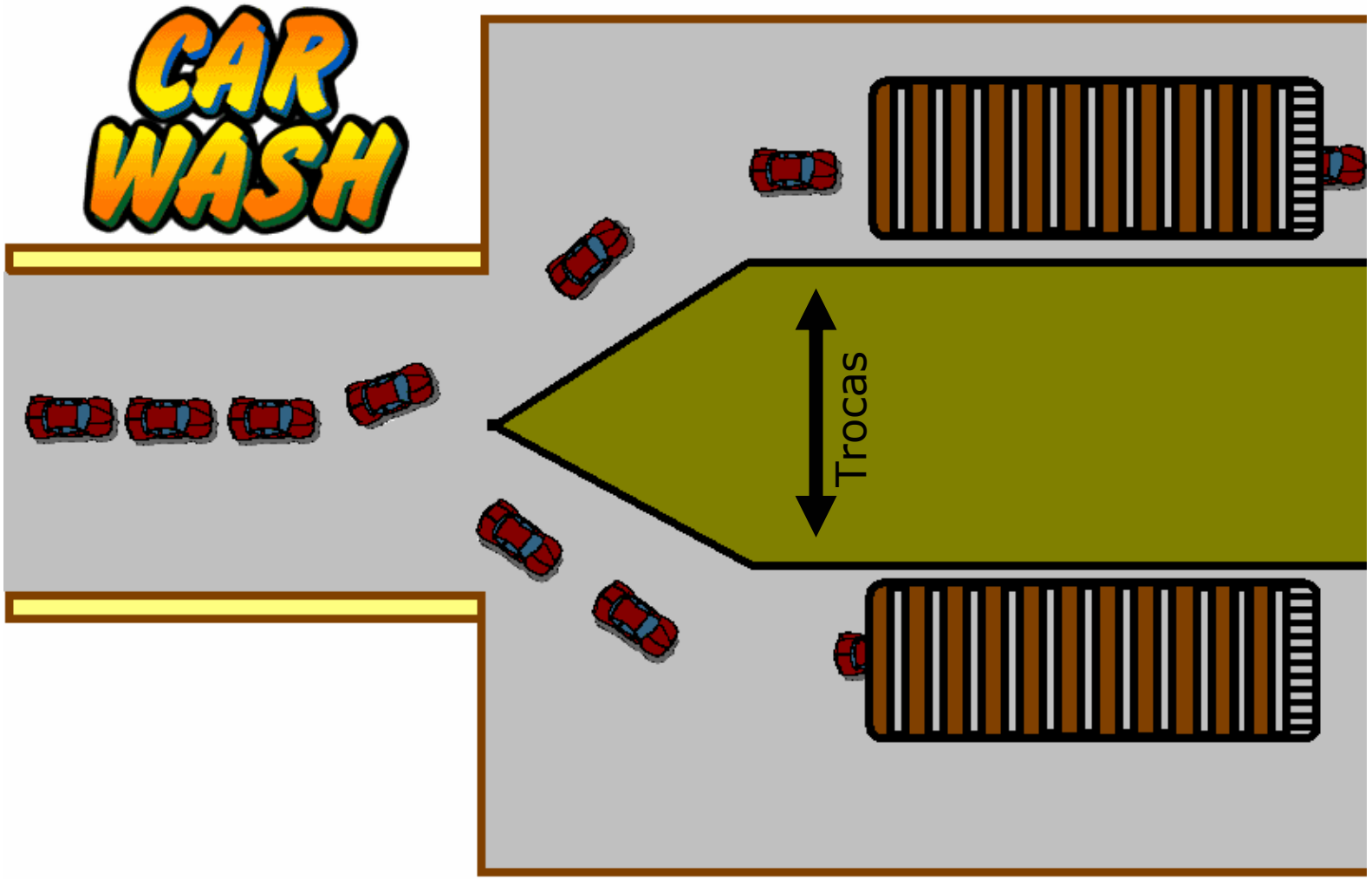
- Programa de simulação
- Aspectos:
  - Coletor de Estatísticas
  - ThreadSafe
  - Trace
  - Precedente
  - Persistência
- Ferramentas
- Conclusões

Conhecendo o Problema

- Desenvolver um programa em uma linguagem de propósito geral, que permita a modelagem e a simulação de um sistema de lavação de carros.

- Descrição do Sistema: As entidades chegam ao sistema e entram em uma das filas para serem atendidas por um servidor. A figura 1 abaixo permite visualizar este sistema. As entidades se dividem em dois tipos: 1 e 2. As do tipo 1 são atendidas pelo Server 1 e as do tipo 2 pelo Server 2.

Ilustrando Melhor o Exercício



Minúcias do Problema

- As entidades entram no sistema e devem escolher para qual cabine irão seguir. Existem então dois tipos de entidades, a do tipo 1 e tipo 2.
- Cada servidor é especializado para melhor servir um tipo de entidade. O servidor 1 serve melhor entidades do tipo 1 e o servidor do tipo 2, entidades do tipo 2;
- O sistema tem apresentado algumas falhas no seu funcionamento, gerando alguns problemas para os proprietários. O que acontece é que as vezes algum dos servidores param. Com isto os carros que estavam na fila deste servidor migram para o final da fila do outro servidor, se este estiver funcionando.
- As entidades que chegam durante o tempo em que o servidor do seu tipo está em falha não são atendidas pelo servidor, sendo imediatamente descartadas do sistema e contabilizadas como cliente perdidos pelos proprietários;

Nominando

- Servidor → Cada uma das cabines de lavação é um servidor;
- Entidade → cada um dos carros que entra no sistema é uma entidade;
- T.E.C. → É o tempo entre chegadas que pode ser uma distribuição de probabilidade ou uma constante;
- T.S. → É o tempo de serviço, ou seja, o tempo que o servidor leva para processar cada uma das entidades. Cada servidor possui o seu próprio TS;
- T.E.F → Tempo entre falhas, que é o tempo entre uma falha e outra;
- T.F. → É o tempo de falha, ou seja, o tempo em que o servidor não poderá servir a nenhuma entidade do sistema.

Onde entram os aspectos???

- O sistema foi construido para que se pudesse fazer a simulação e com isso visualizar os acontecimentos um a um no servidor e no sistema como um todo.
- Porém não foram desenvolvidos qualquer tipo de resultados numéricos com este programa;
- Pensou-se então em fazer com que ele gerasse um relatório contendo algumas informações para que se pudesse mensurar o volume de carga no sistema bem como a desenvoltura dos servidores e o tempo e quantidade de entidades adentrando o sistema.

Estatísticas

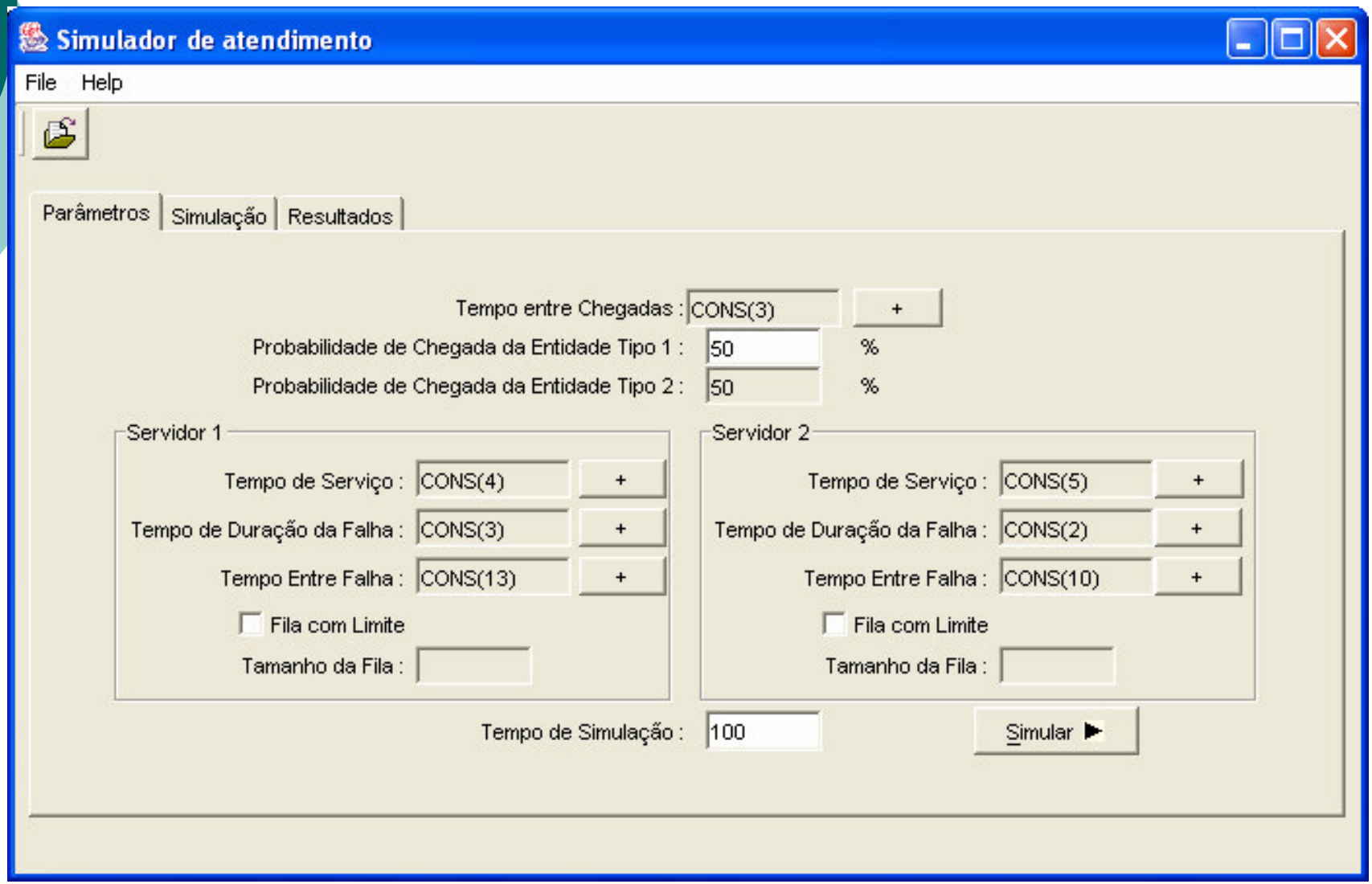
As estatísticas coletadas foram estas:

- Número total de entidades no sistema;
- Tempo total de entidades no sistema;
- Tempo máximo, médio e mínimo de uma entidade no sistema;
- Tempo máximo, médio e mínimo de ocupação dos servidores;
- Tempo máximo, médio e mínimo de falhas e entre falhas nos servidores;

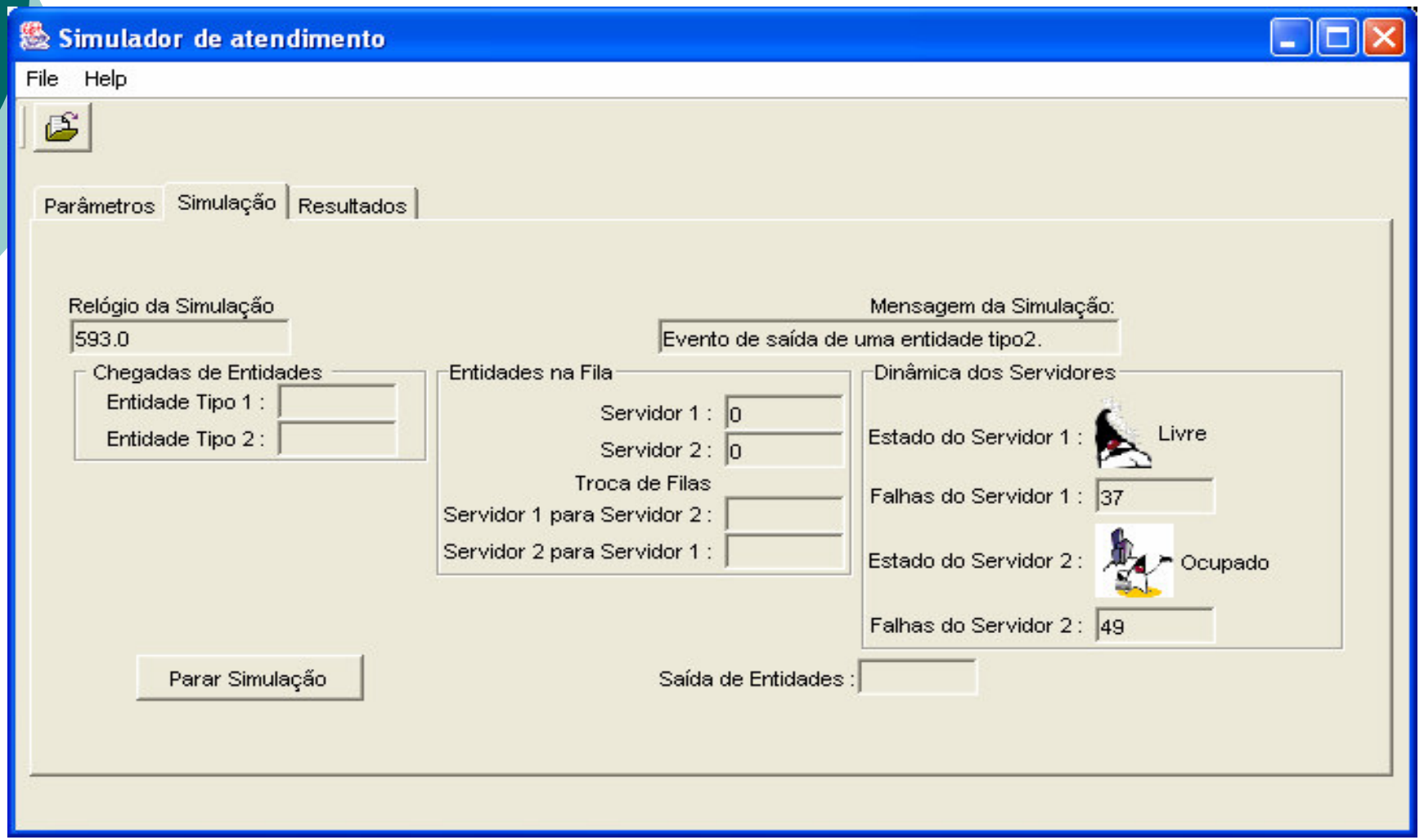
Solução!!!

- Optou-se então por usar aspectos para construir tais relatórios uma vez que:
  - Precisava-se de uma aplicação real para a orientação a aspectos;
  - O código não necessitaria de alterações;
  - A coleta de estatísticas pode ser facilmente modularizável pois não se queria fazer um espalhamento desordenado de código dentro do sistema;
  - O código original foi produzido em JAVA e esta linguagem dispões de uma ótima extensão para a orientação a aspectos, o AspectJ;

Tela Inicial do Simulador



Tela da Simulação



Após Aplicação dos Aspectos

- Criou-se o aspecto ColetorEstatisticas. Com ele foi capaz de capturar-se todas os dados pertinentes para uma boa análise do sistema;
- Criou-se ainda um impressor que gera um relatório final com as estatísticas em .HTML.
- O resultado final foi o esperado, ou seja, conseguiu-se realizar a análise conforme os dados gerados.

Impressor do Relatório

```
package projetofinal;

import java.io.*;
```

```
public class HTMLw {
	private FileWriter out = null;

	public HTMLw(String nome){
		try {
			out = new FileWriter(nome);
		}catch(IOException _ioex){
			System.out.println(_ioex.getMessage());
		}
	}

	public void escreve(String texto){
		try {
			out.write(texto);
			out.flush();
		}catch(IOException _ioex){
			System.out.println(_ioex.getMessage());
		}
	}

	public void fecha(){
		try {
			this.out.close();
		} catch (IOException _ioex) {
			System.out.println(_ioex.getMessage());
		}
	}
}
```

O Aspecto *ColetorEstatistica*

```
pointcut saida(Item evento) :
call( void projetofinal.Simulator.saida(Item)) && args(evento);
```

O *pointcut saida* é responsável por capturar as estatísticas de saídas das entidades do sistema bem como o término dos processos nos servidores. A cláusula *args(evento)* é usada para se ter acesso aos argumentos da chamada.

```
after(Item evento) : saida(evento) {
    int servidorIndice = evento.getServidor().getTipo()-1;
    Entidade ent_atual;
//...
}
```

Usado em conjunto com o *advice after* para que seja executado somente ao final do código original. As duas variáveis disposta aqui servem para se ter acesso ao servidor e entidade atuais, dos quais serão extraídos os dados a serem contabilizados.

O Aspecto *ColetorEstatistica* (2)

```
pointcut falha(Item evento) :
call( void projetofinal.Simulator.falha(Item)) && args(evento);
```

O *pointcut falha* é responsável por capturar o momento inicial dos eventos que geram as falhas nos servidores do sistema. Também aqui usou-se a cláusula *args(evento)* para se ter acesso aos argumentos da chamada.

```
after(Item evento) : falha(evento){
    int servidorIndice = evento.getServidor().getTipo()-1;
//...
}
```

Usado em conjunto com o *advice after* para que seja executado somente ao final do código original. A variável disposta aqui serve para se ter acesso ao servidor do qual será extraído os dados das falhas a serem contabilizados.

O Aspecto *ColetorEstatistica* (3)

```
pointcut retornofalha(Item evento) :
call( void projetofinal.Simulator.retornoFalha(Item)) &&
args(evento);
```

O *pointcut retornofalha* é responsável por capturar o término dos eventos geradores das falhas nos servidores do sistema. Usou-se a cláusula *args(evento)* para se ter acesso aos argumentos da chamada.

Do uso em conjunto dos dados gerados neste pointcut e os dados gerados no *pointcut* anterior (*falha*) pode-se capturar as estatísticas necessárias aos cálculos das falhas nos servidores.

```
after(Item evento) : retornofalha(evento) {
int servidorIndice = evento.getServidor().getTipo()-1;
//...
}
```

Usado em conjunto com o *advice after* para que seja executado somente ao final do código original. A variável disposta aqui serve para se ter acesso ao servidor do qual será extraído os dados das falhas a serem contabilizados.

O Aspecto *ColetorEstatistica* (4)

```
pointcut execucao() :
execution(public void projetofinal.Simulator.run(..));
```

O *pointcut execucao* é responsável pela "marcação" do ponto de corte do método principal de execução. Usado em conjunto com o *advice before* ele é usado para inicializar as variáveis de controle estatístico, criar o novo arquivo de saída e escrever o seu cabeçalho.

```
before() : execucao() {
    ((Simulator)thisJoinPoint.getThis()).setOutHTML(new
    HTMLw("saida.html"));
    ((Simulator)thisJoinPoint.getThis()).getOutHTML().escreve(
    cabecalho());
    iniciaVariaveis();
}
```

O Aspecto *ColetorEstatistica* (5)

Já quando usado em conjunto com a o *advice after* ele tem a função de escrever os dados no arquivo de saída bem como escrever o final da tabela .html criada;

```
after() : execucao() {
    HTMLw htmlo = (Simulator)thisJoinPoint.getThis()).getOutHTML();
    htmlo.escreve(montaLinha("Tempo do Servidor 1 ocupado :",
    Double.toString(Funcoes.round(tempoOcupado[0]))));
//...
    htmlo.escreve("</table></body></html>");
    htmlo.fecha();
}
```

Resultados

- Com a aplicação da orientação a aspectos no nosso programa, conseguimos alcançar com grande êxito o que nos propomos fazer, que era coletar as estatísticas para que pudéssemos mensurar o nosso sistema com a finalidade de melhorar ao máximo o atendimento dos servidores às entidades;
- Já esperávamos conseguir alcançar tal resultado graças a experiência anterior resultante da aplicação de aspectos no trabalho de desenvolvimento dos containers;
- Com os aspectos conseguimos evitar mudanças no programa fonte original, bem como evitamos também um grande acúmulo de códigos não pertinentes ao simulador, separando-os em um aspectos que futurmante também pode ser re-utilizado;

ThreadSafeAspect

- Permite executar automaticamente replicações da simulação;
- Interação com o usuário no questionamento da quantidade de replicações;

ThreadSafeAspect(2)

- Pontos de corte:

  - **pointcut inicia() :**
    execution(public void projetofinal.Simulator.iniciarSimulacao());
    → Captura a execução do método iniciarSimulacao;
  - **pointcut iniciaCall() :**
    call(public void projetofinal.Simulator.iniciarSimulacao());
    → Captura a chamada do método iniciarSimulacao;
  - **pointcut atribuihtml(HTMLw arq) :**
    execution(public void projetofinal.Simulator.setOutHTML(HTMLw)) && args(arq) ;
    → Captura a execução do método setOutHTML e seu parâmetro;
  - **pointcut construtor(Tela tela) :**
    initialization (projetofinal.Simulator.new(Tela))&& args(tela);
    → Captura a inicialização (Construtor) da classe Simulator e seu parâmetro
  - **pointcut execucao() :**
    execution(public void projetofinal.Simulator.run(..));
    → Captura a execução do método run;

TraceAspect e PrecedenteAspect

- TraceAspect: Realiza a monitoração dos eventos da simulação, armazenando-os em um arquivo texto com o uso do pacote java.util.logging;
- PrecedenteAspect: Aspecto responsável pela prioridade de aplicação dos aspectos.
  - **declare precedence**:
    PrecedenteAspect,ColetorEstatisticas,
    ThreadSafeAspect,TraceAspect,*;

PersistenciaAspect

Voltando aos TAD, esse aspecto implementa a persistência da lista duplamente encadeada em arquivo;

Comentários Sobre as Ferramentas

- Eclipse
  - Ótima ferramenta para desenvolvimento de aplicativos JAVA;
  - Com a possibilidade de plug-ins foi capaz de "adaptar-se" a necessidade de produzir orientação a aspectos com notável qualidade;
- JBuilder
  - Plugin um pouco imaturo, principalmente na edição do arquivo de configuração de construção, utilizado na compilação dos códigos.
  - Mas tem um diferencial do Eclipse por permitir completação de código java.

Conclusões

- Constatou-se que com o uso dos aspectos poupou-se um enorme esforço, uma vez que não foi necessário alterar um linha do código fonte;
- A priori, no programa original, existia um projeto de coletor de estatística. Porém este coletor estava totalmente espalhado no código e sendo impossível a sua futura reutilização separadamente ao simulador;
- Já a parte de threadsafe tornou-se com passar do tempo um busca pessoal para aprimoramento do simulador, uma vez que o programa inicial não previa múltiplas replicações automaticamente. Com o aspecto implementado, conseguiu-se realizar a tarefa com a garantia de ter-se os dados das iterações coerentemente separados em arquivos, previamente identificado pelo ID de cada iteração;
- O aspecto resposável pela persistência nos TAD surgiu como complemento ao trabalho realizado na primeira parte da disciplina, mais a título de curiosidade e desenvoltura. A persistência de escrita em arquivo é realizada com sucesso com o aspecto gerado;
- O aspecto Precedente foi criado com o único intuito de realizar a ordenção da aplicação dos aspectos em cima do código fonte.