# Java ME
## Utilizando Floggy para Persistência de dados

CARLOS EDUARDO CARVALHO DANTAS

**Utilize o poder do Floggy, um Framework criado por brasileiros que veio para substituir o RMS em Java ME**

Dentro do Desenvolvimento de Aplicações em Java ME, um dos maiores problemas que encontramos atualmente é com relação à armazenagem ou persistência dos dados. Isto porque, os sistemas operacionais dos dispositivos fornecem recursos simples para persistência de dados e os poucos bancos de dados existentes são proprietários e disponíveis somente para determinados tipos de dispositivos.
Atualmente a persistência em Java ME é feita com RMS (Record Management System), que usa o Record Store para gerenciamento de registros em um array de bytes com um identificador único para cada registro. Esta maneira de persistir os dados torna-se um trabalho complexo para o programador, já que cabe ao mesmo a responsabilidade de gerenciar as posições do um array além dos separadores entre os campos da tabela, tratamento de exceções de IO, etc.. .
Com todos os problemas apresentados anteriormente, torna-se necessária uma ferramenta de nível mais alto para o desenvolvedor, que abstrai todos os detalhes de memória. O framework Floggy fornece exatamente isso.

### O Framework Floggy

O Framework Floggy visa abstrair os detalhes de persistência do programador, ficando este responsável apenas pela montagem da lógica da aplicação, ou seja, ao invés de gerenciar o conteúdo das tabelas por array de dados, a lógica de gerenciamento será como um bean em J2EE, aonde basta criar as classes modelo, setar o valor de cada atributo e persistir. Isto facilita bastante o trabalho do programador, já que cabe ao mesmo pensar nas tabelas do banco como objetos de uma classe, não como array de bytes.

### Download e Instalação

Para fazer a instalação do Floggy, efetue o download do arquivo floggy.zip no site http://sourceforge.net/projects/floggy. Após descompactá-lo, verá uma série de informações, inclusive tutoriais em português, bibliotecas do framework e o código fonte, caso deseje implementar alguma funcionalidade a mais, específica para o seu negócio. O Floggy vem com o arquivo org-floggyuml-floggyumlplugin.nbm (daqui em diante será chamado de “arquivo .nbm”) para fazer a instalação do plugin FloggyUML no NetBeans e dois arquivos importantes que fazem parte do FrameWork que são floggy-framework-SNAPSHOT.jar (daqui em diante será chamado de “arquivo .jar”) que é a biblioteca do FrameWork Floggy e floggy-compiler-Java ME-midp2-1.0.zip (daqui em diante será chamado de “arquivo .zip”) que é responsável pela compilação do Floggy. Lembrando que, o Floggy pode ser utilizado em qualquer IDE para Java, a única diferença é que, para o NetBeans, foi desenvolvido o plugin FloggyUML que se torna bastante útil no aspecto da criação dos diagramas de classe UML para geração das classes Java correspondentes ao teu negócio. Para conferir como se utiliza o Floggy na IDE Eclipse, vá no quadro “Floggy com o Eclipse”.
Para instalar o FloggyUML no NetBeans, clique na aba Tools/UpdateCenter. Selecione Install Manually Downloaded Modules. Clique em add e adicione o arquivo .nbm já citado anteriormente. Clique em next, next, accept e o plugin será instalado. Depois basta selecionar a opção plug-in FloggyUMLPlugin e reiniciar a IDE.

Crie um novo projeto no NetBeans e vamos instalar os outros dois arquivos, o .jar e o .zip. Copie-os para o diretório raiz do seu projeto, acesse as propriedades do projeto, selecione a opção Libraries & Resources e clique em Add jar/zip. Escolha o arquivo .jar e clique em abrir conforme pode ser visto na Figura 1.

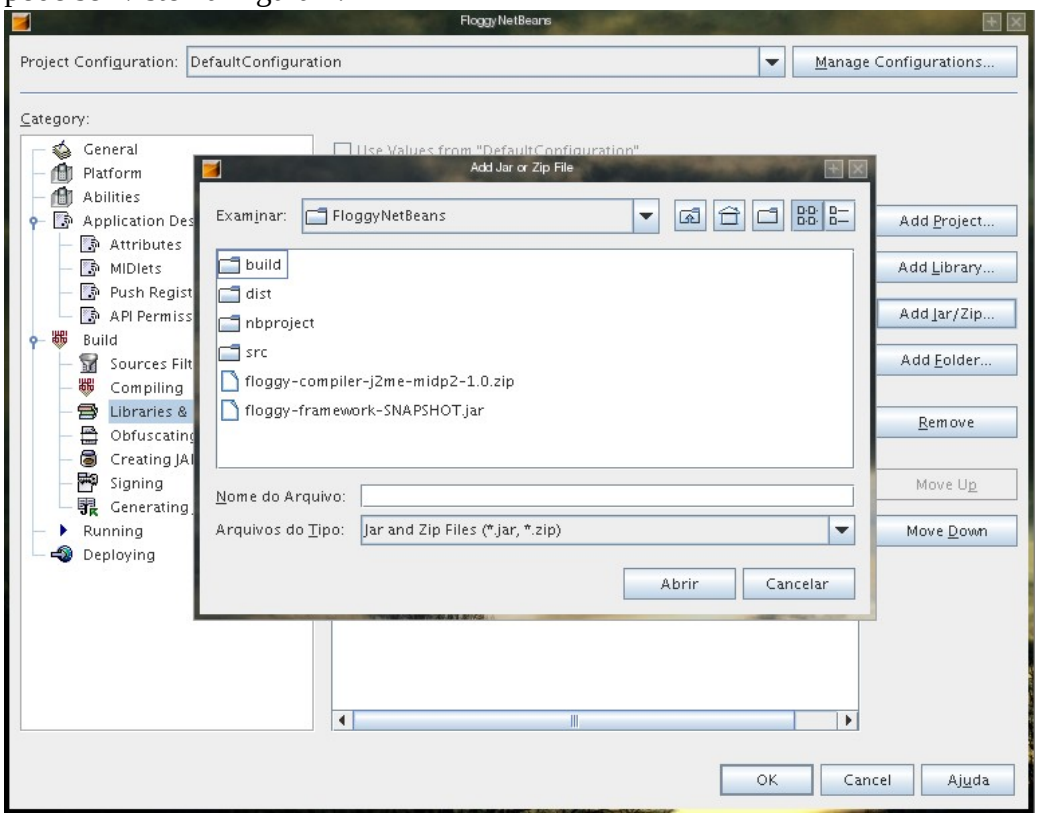


**Figura 1**. Adicionando a blblioteca floggy-framework-SNAPSHOT.jar no projeto.

Para referenciar o arquivo .zip no projeto, basta configurar o arquivo build.xml da sua aplicação. O arquivo build.xml foi gerado automaticamente pelo NetBeans na criação do projeto. Para acessá-lo, basta selecionar a guia Files e conferir os arquivos do projeto. Abra-o e adicione o código da Listagem 1 no final do arquivo build.xml, antes do fechamento da tag project. O código é um script ant simples, aonde a tag taskdef define quem irá compilar o projeto, no caso está sendo referenciado o arquivo .zip em floggy-compiler, em seguida a tag floggy-compiler chama o compilador passando o local dos arquivos compilados (inputfile) e os arquivos de saída (outputfile). Com isso, o projeto já está pronto para desenvolvimento da aplicação com Floggy.

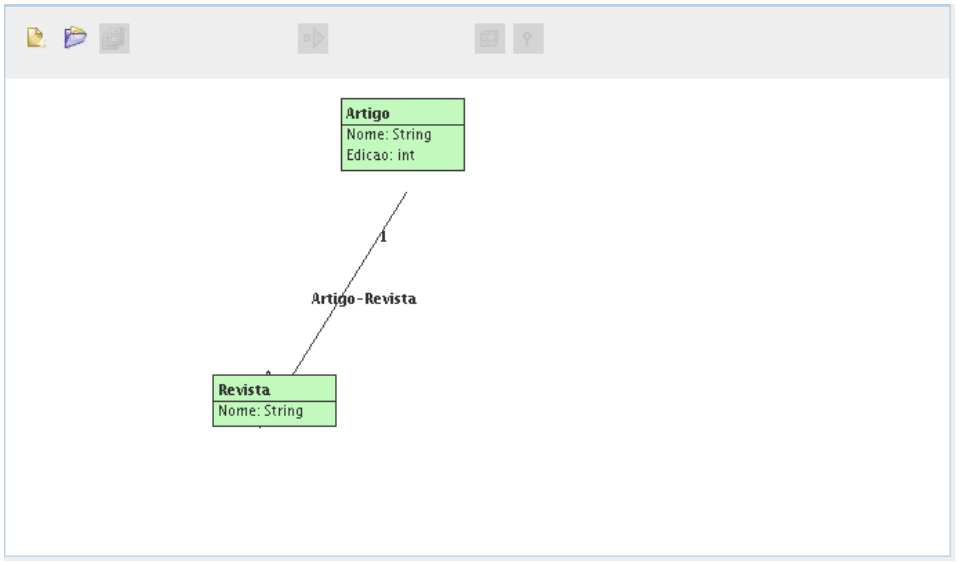

**Figura 2**. Diagrama criado no FloggyUML

## Utilização do Plug-in FloggyUML no NetBeans

Dentro do projeto, vamos criar o modelo de dados. O aplicativo é basicamente um cadastro de artigos de revistas, contendo duas tabelas: Artigo e Revista. Revista contém apenas o campo nome enquanto artigo contém os campos Nome, Edição e uma referência para a tabela Revista.
A janela do plugin FloggyUML está em Window – Open DiagramEditor Window. Como dito anteriormente, o FloggyUML é um plugin bastante útil na geração das classes Java que servirão de modelo para as tabelas. Basta criar os diagramas com as relações entre as classes e exportar e todo o modelo das tabelas está pronto. O funcionamento do FloggyUML é bastante simples e intuitivo, mas vamos seguir um passo-a-passo para ficar de fácil entendimento. De cara, vemos dois botões habilitados. Clique na opção New Diagram, a primeira opção é para apontar o diretório raiz do projeto e a segunda é para especificar o nome do package em que as classes java ficarão. O segundo campo é opcional. Feito isso, crie uma nova classe clicando botão “Add new class” e arrastando até a área branca. Feito isso basta renomear a classe e clicar com o botão direito para adicionar atributos. Crie as duas classes conforme a Figura 2. Para criar associação entre as classes artigo e autor, clique no botão “Add new Association”, clicando logo em seguida sobre as duas classes que farão parte do relacionamento. Por default é criada uma associação de 1(um) para 1(um) entre as duas classes. Para modificar, basta mudar o valor de alguma das associações para “*” (muitos). Também é possível mudar o nome da associação como desejar. Com o botão “Start Code Generator” são gerados dois arquivos, Revista.java e Artigo.java dentro do projeto. Também pode salvar o diagrama clicando no botão “Save Diagram”, caso precise alterar alguma classe futuramente ou criar novas.
Analisando as classes java criadas, vemos que a novidade fica por conta do método save(). Este método é quem salva o nosso modelo de dados, conforme veremos adiante. O código das classes Artigo e Revista podem ser conferidos nas Listagens 2 e 3 respectivamente.

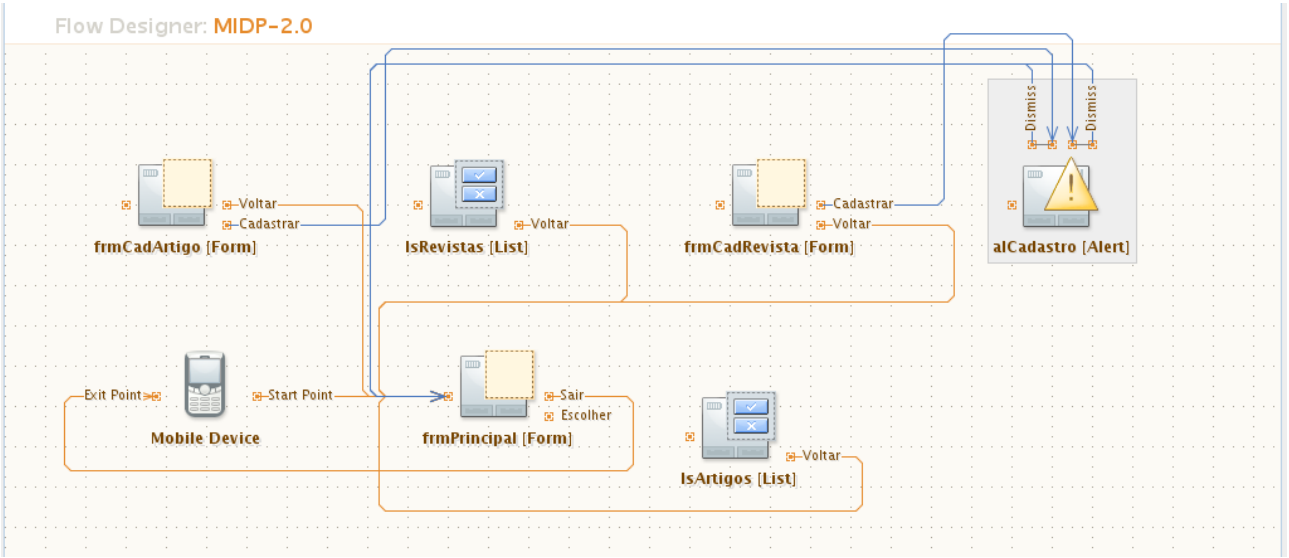


**Figura 3**. Flow Design da aplicação

## Montando a Aplicação

Até aqui, já foi instalado e referenciado o Floggy na aplicação e também foi criado o modelo de dados. Agora precisamos desenvolver as telas no NetBeans, conforme pode ser visualizado na Figura 3. Como o foco do artigo não é ensinar a desenvolver em Java ME, presumimos que o leitor já tenha conhecimentos básicos acerca de desenvolvimento Java ME com NetBeans. Caso não, aconselho o artigo Programação JavaME da edição 46, aonde aborda passo a passo como desenvolver em Java ME no NetBeans. Na Figura 3, podemos ver que a aplicação possui um mobile device, três Forms, sendo que um é o principal, o menu da aplicação e os outros dois são

para Cadastro de Artigo e Revista cada um, dois Lists para listar os dados de Artigo e Revista e um Alert para mostrar mensagens na tela. Quanto aos Commands das telas, apenas o form principal tem command para sair da aplicação. Os outros Forms possuem command para voltar para o form principal e para salvar os dados e os dois Lists possuem command para voltar para o form principal apenas.
Os Screen Design dos forms de Cadastrar Artigo e Cadastrar Revista podem ser visualizados nas Figuras 4 e 5 respectivamente. Em Cadastro de Artigo possuem dois TextFields para Nome e Edição e um ChoiceGroup Popup para Revista. Já Cadastro de Revista possui um TextField apenas para Nome.


**Figura 4.** Screen Design do form de Cadastro de Artigo

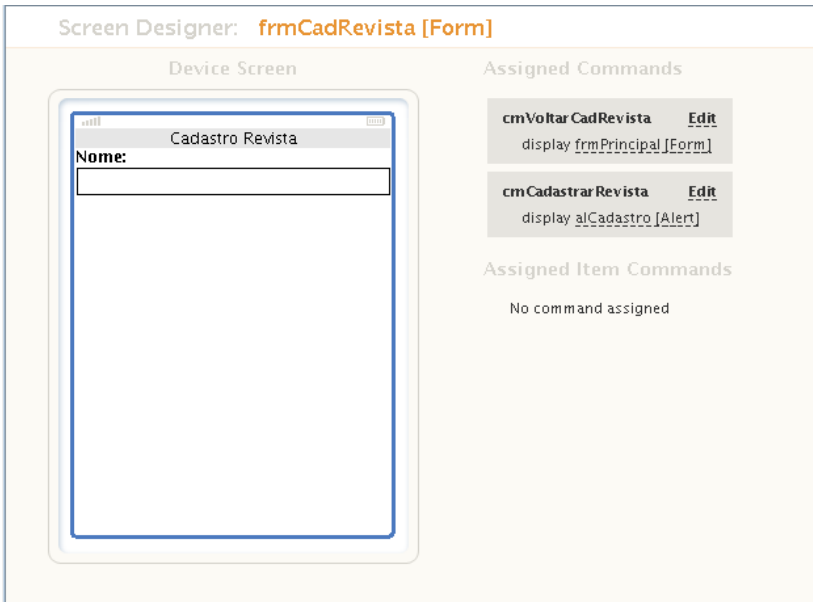


**Figura 5.** Screen Design do form de Cadastro de Revista

## Persistindo os dados

Persistir utilizando Floggy é muito mais fácil do que se imagina. Vamos utilizar como exemplo a classe Revista. Esta classe possui apenas um único atributo chamado Nome. Quando o usuário

clicar no Command de Salvar, o único trabalho que teremos é de instanciar uma nova classe Revista, setar o atributo Nome com o valor preenchido no TextField pelo usuário e chamar o método save() que se encontra dentro da classe Revista. O exemplo está na Listagem 4. Com certeza isto é muito mais fácil e muito mais prático que montar uma String para cada registro novo com separadores entre os campos, como é usado atualmente em RMS. Implemente o código da Listagem 4 dentro da condição do Command Salvar dentro do Form de Cadastro de Revista.
Para o Command Salvar do Form de Cadastro de Artigos, segue-se a mesma lógica, apenas com uma diferença: como o cadastro de artigos possui uma referência à classe Revista, precisa de recuperar a instância de Revista que a String selecionada dentro do ChoiceGroup, por isso que usa-se o método Find aplicando Filter conforme pode ser visto na Listagem 5. A lógica de se utilizar o Filter será explicada na seção Filter e Comparator e os métodos de buscar registros será detalhada na seção a seguir.

## Listando os dados

Se salvar os dados utilizando o RMS já é complicado, imagina buscá-los da base. Para começar, o separador precisa ser um caractere que o usuário nunca poderá usar para digitar algo, caso contrário não saberemos se o caractere em determinada posição da String seria um separador ou parte da String de um campo. Outro detalhe é a manipulação de arquivos que sempre podem gerar exceção. Com o Floggy, a busca de dados é muito simples. Para listar os registros da base, basta implementar o código na Listagem 6. Visualizando o código, ObjectSet tem a finalidade de receber um array de dados através de find ou findAll. find é uma busca que pode ser filtrada ou ordenada ou nenhuma das anteriores, enquanto findAll traz todos os registros sem filtro e sem ordenação. Cada iteração do loop for tratá um registro recuperado da base, e adicionando no List de Artigos.
A lógica do List de Revistas é praticamente igual, basta mudar para Revista.class dentro do findAll e mudar os gets dos campos da classe Artigo para os gets dos campos da classe Revista dentro do loop.

## Filter e Comparator

Filter e Comparator são duas interfaces bastante úteis no Floggy. A semelhança entre estas duas interfaces é que para utilizá-las, precisa-se criar classes separadas implementando estas interfaces. A diferença é que Filter está associado aos filtros de uma consulta enquanto que Comparator está associado quanto à ordenação de uma consulta.
O exemplo de Filter pode ser conferido na Listagem 7. Quando o método find é chamado, este faz um loop interno buscando os registros da tabela passada como parâmetro a cada iteração, entra no método matches para conferir se cada registro do loop passa ou não pelo filtro. Repare que, quando se implementa a interface Filter, a declaração do método matches torna-se obrigatória. Este método tem como parâmetro um objeto do tipo Persistable, que pode ser convertido para a classe passada como parâmetro, podendo filtrar os atributos desejados. O retorno é um boolean que diz se aquele objeto em específico passou ou não no filtro.
A interface Comparator segue uma idéia parecida, já que a classe que for implementar Comparator precisa declarar o método compare passando dois Persistables como parâmetro. A idéia é fazer um cast para a classe modelo desejada e depois comparar o campo que quiser, no caso o método compareTo compara por caracteres alfanuméricos. O código da interface Comparator pode ser conferido na Listagem 8.

## Conclusão

Com o que foi apresentado, podemos concluir que apesar de ser um framework de terceiros, o Floggy é uma ótima opção para persistência em aplicações Java ME, levando para o programador uma solução de alto nível sendo prática, robusta e simples de codificar, podendo substituir a biblioteca padrão RMS que é utilizada atualmente para persistência na maioria das aplicações Java ME.

## Floggy com o Eclipse

Apesar deste artigo dar uma ênfase maior na IDE NetBeans, o Floggy também pode ser utilizado no Eclipse. Boa parte da codificação explicada no artigo pode ser utilizada da mesma forma para o Eclipse. O único aspecto negativo fica por conta do plugin FloggyUML ainda não possuir sua versão para o Eclipse, tendo como conseqüência a necessidade de criar as classes-modelo manualmente, conforme pode ser visto nas Listagens 1 e 2.

Primeiramente, antes de configurar o Floggy no Eclipse, para que o leitor fique por dentro das diferenças, toda a configuração explicada na sessão Download e Instalação, foi referenciada a IDE NetBeans, portanto para o Eclipse, a configuração é totalmente diferente, como por exemplo o script ant da Listagem 1 não se faz necessário no Eclipse, nem referenciar o .jar e .zip explicados na sessão.

A configuração do Floggy para Eclipse é bastante trabalhosa, mas a boa notícia é que os criadores do Floggy já estão trabalhando em um plugin único e definitido para o eclipse. Mas para quem quer usar o que temos atualmente, primeiramente entre os arquivos que podem ser baixados do artigo no site da DevMedia, procure a pasta floggy-persistence-1.0 para o diretório raiz do projeto. Feito isso, acesse as propriedades do projeto e adicione o arquivo floggy-persistence-framework.jar que está dentro da pasta lib ao classpath da sua aplicação e na opção de exportar, marque-o.

Em seguida, é necessário adicionar a ferramenta que irá enxergar o código de persistência, conhecida como Weaver. Para isto, selecione a opção Builders em propriedades do projeto e crie uma nova entrada clicando em New, Ant Build.

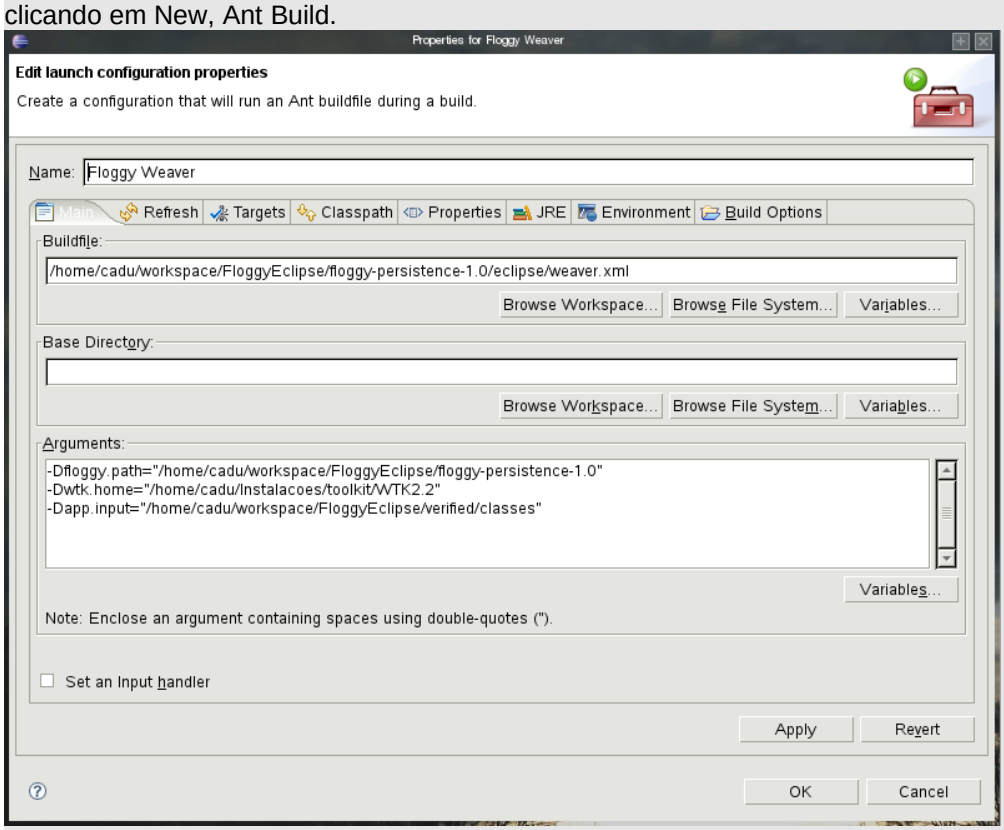


**Figura Q1**. Exemplo preenchimento dos campos para configurar o Weaver.

Agora preencha os campos conforme a Figura Q1. Lembre-se de alterar os parâmetros para trabalhar com as suas configurações.

- floggy.path – diretório aonde o floggy foi instalado (descompactado);
- wtk.home – diretório de instalação do Sun Wireless Toolkit;
- app.input – diretório aonde os arquivos .class estão sendo gerados pelo Eclipse.

Concluindo, dentro da opção builders aparecerá o Floggy Weaver. Selecione-o e clique em Up para que ele fique entre os outros dois builders, o Java Builder e o CLDC Preverifier.

Bem, para quem esteja utilizando no eclipse os imports dos códigos apresentados aqui, especialmente das Listagens, existem algumas ressalvas com relação ao Eclipse:

- No Eclipse, o caminho dos pacotes dos imports é diferente, portando utilizando um simples Ctrl + space as classes que estiverem apresentando erro acharão o caminho correto;
- No Eclipse o método findAll simplesmente não existe! Para simulá-lo, basta utilizar find(nomedaclasse.class,null,null) que terá a mesma funcionalidade de findAll;
- A classe ObjectSet ao invés de oferecer o método count(), oferece o método size().

**Carlos Eduardo Carvalho Dantas** (carlos_eduardo@digitalhal.com.br) é Bacharel em Engenharia de Computação pela FEIT-UEMG e pós graduando em Desenvolvimento Java pela UNITRI. Atualmente trabalha como Analista/Desenvolvedor Java para a empresa Digital Hall em Uberlândia - MG.

## Links

***floggy.org***
Sobre o Floggy

**Listagem 1**. *Configuração no arquivo build.xml do projeto*

```
    <property name="path.floggy.CompilerTask" location="floggy-compiler-J2ME-midp2-1.0.zip" />
    <property name="path.floggy.framework" location="floggy-framework-SNAPSHOT.jar" />

    <taskdef name="floggy-compiler" classname="net.sourceforge.floggy.CompilerTask"
       classpath="${path.floggy.CompilerTask}" />

    <target name="post-compile">
       <floggy-compiler
          classpath="${path.floggy.CompilerTask}${path.separator}${path.floggy-framework}"
          inputfile="${basedir}/build/compiled"
          outputfile="${basedir}/build/compiled" />
    </target>
```

**Listagem 2**. *Exemplo de classe-modelo gerada pelo FloggyUML Artigo.java*

```
import net.sourceforge.floggy.FloggyException;
import net.sourceforge.floggy.Persistable;
import net.sourceforge.floggy.PersistableManager;

public class Artigo implements Persistable{

        private String Nome;

        private int Edicao;

        private Revista revista;

        public String getNome(){
                return this.Nome;
        }
        public void setNome(String Nome){
                this.Nome = Nome;
        }
        public int getEdicao(){
                return this.Edicao;
        }
        public void setEdicao(int Edicao){
                this.Edicao = Edicao;
        }
        public Revista getRevista(){
                return this.revista;
        }
        public void setRevista(Revista revista){
                this.revista = revista;
        }

        public void save(){
                try{
                        PersistableManager.getInstance().save(this);
                } catch (FloggyException ex) {
                        ex.printStackTrace();
                }
        }
}
```

**Listagem 3**. *Exemplo de classe-modelo gerada pelo FloggyUML Revista.java*

```
import net.sourceforge.floggy.FloggyException;
import net.sourceforge.floggy.Persistable;
import net.sourceforge.floggy.PersistableManager;

public class Revista implements Persistable{

        private String Nome;
```

```
	public String getNome(){
		return this.Nome;
	}
	public void setNome(String Nome){
		this.Nome = Nome;
	}

	public void save(){
		try{
			PersistableManager.getInstance().save(this);
		} catch (FloggyException ex) {
			ex.printStackTrace();
		}
	}
}
```

**Listagem 4.** *Código implementado no command Salvar da tela de Cadastro de Revistas*

```
  if (command == cmCadastrarRevista) {
        Revista revista = new Revista();
        revista.setNome(tfNome.getString());
        revista.save();
        getDisplay().setCurrent(get_alCadastro(), get_frmPrincipal());
  }
```

**Listagem 5.** *Código implementado no command Salvar da tela de Cadastro de Artigos*

```
                Artigo artigo = new Artigo();
                artigo.setNome(tfNomeArtigo.getString());
                artigo.setEdicao(Integer.parseInt(tfEdicao.getString()));
                ObjectSet os = null;
                try {
                   os = PersistableManager.getInstance().find(Revista.class,new
FiltroRevista(cgRevistas.getString(cgRevistas.getSelectedIndex())),null);
                   artigo.setRevista((Revista)os.get(0));
                } catch (FloggyException ex) {
                         ex.printStackTrace();
                         artigo.setRevista(null);
                }

                artigo.save();
                getDisplay().setCurrent(get_alCadastro(), get_frmPrincipal());
```

***Listagem 6.*** *Código implementado no Command Listar da tela de listar Artigos*

```
                ObjectSet os = null;
                get_lsArtigos().deleteAll();
                try {
                     os = PersistableManager.getInstance().findAll(Artigo.class);

                     for (int i = 0; i < os.count(); i++) {
                         Artigo artigo = (Artigo)os.get(i);
                         get_lsArtigos().append(artigo.getNome()+ " - "+ artigo.getEdicao() + " - "
+ artigo.getRevista().getNome() +  "\n", null);
                     }
                } catch (FloggyException ex) {
                     get_lsArtigos().append("Erro: List Record", null);
                }
                getDisplay().setCurrent(get_lsArtigos());
```

***Listagem 7.*** *Código implementado na classe FilterRevista.java*

```
public class FiltroRevista implements Filter {
    private String nome;
    public FiltroRevista(String nome) {
        this.nome = nome;
    }
    public boolean matches(Persistable objeto) {
        Revista r = (Revista) objeto;
        return r.getNome().equals(nome);
    }
}
```

***Listagem 8.*** *Código implementado na classe ComparatorRevista.java*

```
public class ComparatorRevista implements Comparator {

   public int compare(Persistable data1, Persistable data2) {
        Revista revista1 = (Revista)data1;
        Revista revista2 = (Revista)data2;
        String name1 = revista1.getNome();
        String name2 = revista2.getNome();
        int result = name1.compareTo(name2);
        if (result == 0) {
            return RecordComparator.EQUIVALENT;
        } else if (result < 0) {
            return RecordComparator.PRECEDES;
        } else return RecordComparator.FOLLOWS;
    }
}
```