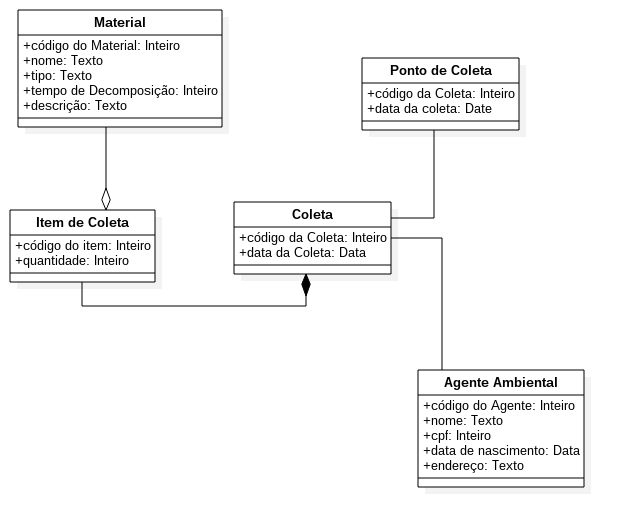
O caso de uso escolhido para a implementação das aplicações Java é baseado em um sistema de coletas. Um agente ambiental realiza uma coleta em conjunto com um ponto de coleta e a data da coleta fica registrada. Para cada coleta podem existir um ou vários Itens de Coleta onde cada Item de Coleta possui um Material e sua respectiva quantidade.

**Diagrama de Classes**

  
Figura x : Diagrama de classe de Coleta

**MySQL com jdbc**

Para início do desenvolvimento dos experimentos, a tecnologia escolhida foi o banco de dados MySQL utilizando o conector JDBC para aplicações Java.

O tema escolhido para o desenvolvimento das aplicações foi um gerenciador de coleta de lixo, onde existe um coletador, um ponto de coleta, os materias que serão coletados pelo coletador, uma coleta que reúne o coletador, seu ponto de coleta além de Itens de Coleta que registram a quantidade do material coletado e a coleta a que pertencem.

Para a criação da aplicação Java com o sistema gerenciador de banco de dados MySQL, a primeira coisa a se fazer é criar as tabelas do banco de dados Coleta.

A tabela de Agente Ambiental possui os seguintes campos: código do agente, nome, cpf, data de nascimento e endereço. A tabela de Ponto de Coleta possui os campos: código do Ponto, nome, endereço, cidade, estado e data de retirada. A tabela de Material possui os atributos: código do material, nome, tipo, tempo de decomposição e descrição. Os atributos presentes na tabela de Coleta são: código da coleta, código do agente, código do ponto e data da coleta. Já a tabela de Itens de Coleta possui os atributos: código do item, código do material, código da coleta e a quantidade de materias.

A tarefa após fazer a criação do banco de dados, é realizar a conexão com o SGBD MySQL. Para isso é necessário baixar um conector e adicioná-lo como biblioteca da aplicação Java. A criação de classes Java para cada tabela do banco de dados se faz necessária devido ao fato de os atributos da classe corresponderem com campos das tabelas. Sendo assim, uma classe para cada tabela do SGBD deve ser criada e os métodos CRUD devem ser implementados em cada uma dessas classes.

O método Insert foi criado em cada classe com o propósito de permitir que objetos de cada classe sejam persistidos no SGBD MySQL com a ajuda do conector. Já o método Update tem o intuito de fazer com os objetos já persistidos no SGBD sejam alterados, dessa maneira se algum dado tenha sido persistido de maneira errônea, é possível alterar esse dado e persisti-lo novamente.

Para fazer a remoção de objetos persistidos, o método Delete foi implementado, o método Select permite que os objetos sejam recuperados para posterior manipulação pelo usuário ou desenvolvedor. O método Select vem a ser um dos métodos mais utilizados pois com ele é possível recuperar os objetos e mostrá-los da maneira que for mais indicada para cada caso. Cada aplicação Java pode ter uma interface gráfica, por meio dessa interface os ususários da aplicação podem interagir com o sistema e realizar as operações CRUD.

Nesse experimento, uma única interface gráfica foi desenvolvida e utilizada em todas as versões do sistema. Para realizar as operações de CRUD na aplicação Java, os eventos foram criados e as operações CRUD criadas manualmente nas classes Java foram invocadas para cumprirem seu propósito.

**Versão Hibernate**

Na criação da aplicação Java utilizando o *framework* Hibernate, não é necessário a criação de tabelas no SQDB MySQL como no experimento do MySQL e JDBC. Sendo assim, é necessário unicamente criar o banco de dados da aplicação Java, posteriormente importar o conector JDBC como biblioteca desse projeto e criar as classes de entidade. Deve ser criado uma classe de entidade para cada futura tabela, pois o Hibernate tem a tarefa de automatizar a conexão com o SGBD, dessa maneira, a tarefa do desenvolvedor é de apenas criar as classes de entidade com os atributos. Os atributos vão se tornar campos das tabelas no SGBD, devem ser definidos também alguns atributos de relacionamento, para demonstrar como cada classe de entidade está relacionada com outra.

Depois da criação das classes de entidade, é necessário a criação de um objeto *Controller.* Esse objeto Controller faz a criação de vários outros objetos *Controller,* um objeto Controller para cada Classe de entidade. Esses objetos são responsáveis por todas as operações CRUD de cada classe, funcionalidade que salva muito tempo do desenvolvedor. Faz-se necessário a criação de uma Unidade de persistência para a aplicação, nessa Unidade de persistência todos os objetos *Controller* são adicionados e recebem a capacidade de persistir seus objetos das Classes de Entidade no SGBD.

A mesma interface gráfica foi utilizada para a interação com o usuário da aplicação, sendo assim, os eventos foram tratados e as operações CRUD presentes nas objetos *Controller* foram usadas.

**Versão Caché Intersystems**

No desenvolvimento da versão utilizando o Banco de dados Orientado a Objetos Caché Intersystems, a primeira etapa a se fazer foi a criação das classes *proxies* na ferramenta *Studio* do cliente de banco de dados Caché. Para a criação dessas classes *proxies*, é preciso colocar todos os atributos que cada classe terá bem como seus respectivos relacionamentos. Depois que todas as classes foram criadas e compiladas, foi preciso adicionar o conector JDBC na pasta de bibliotecas da aplicação Java. A próxima etapa realizada foi a execução de uma projeção dentro da pasta fonte da aplicação Java. Essa projeção faz com que todas as classes *proxies* fiquem disponívveis para a utilização do desenvolvedor dentro da aplicação Java. Nessas classes *proxies* estão inclusas todas as operações CRUD além de vários outros métodos bem úteis.

Tendo realizado esta etapa, faz-se necessário a manipulação dos dados informados pelo usuário da aplicação Java, para isso, a mesma interface gráfica foi utilizada e os eventos foram todos criados com o suporte das operções presenetes dentro das classes *proxies*.

**Versão MongoDB**

Para a aplicação Java utilizando o banco de dados NoSQL MongoDB, foram necessários alguns passos diferentes das versões anteriores pois o conceito NoSQL tem algumas diferenças em relação aos bancos de dados relacionais. Bancos de dados NoSQL são baseado em documentos, cada documento pode conter outros documentos. Esses documentos ficam armazenados em *collections,* que são equivaleantes a tabelas dos bancos relacionais.

Um banco de dados foi criado para o armazenamento das *collections*, optou-se por criar as *collections* dentro da ferramenta *opensource* chamada *RoboMongo* mas essas *collections* poderiam ter sido criadas dentro da aplicação Java diretamente. Foi necessário a criação de todas as classes Java com seus respectivos atributos. Dentro de cada classe Java, foram criados todos os métodos CRUD para que possam ser acessados pela interface gráfica. Os eventos gerados pela interface gráfica foram manipulados utilizando os métodos CRUD das classes Java criados anteriormente. Segue abaixo uma tabela demonstrando a quantidade de tempo gasto com cada versão da aplicação Java

|  |  |
| --- | --- |
| Banco de Dados | Tempo gasto |
| Caché Intersystems | 76,52 minutos |
| MongoDB | 136,48 minutos |
| Hibernate | 126,03 minutos |
| MySQL com JDBC | 184,81 mintos |

As linhas de código que foram escritas também foram contabilizadas e seguem na tabela abaixo.

|  |  |
| --- | --- |
| Banco de Dados | Linhas de Código |
| Caché Intersystems | 633 |
| MongoDB | 721 |
| Hibernate | 607 |
| MySQL com JDBC | 1115 |