Nome: Leon Kacowicz Matricula: 0511148
Relatório do Trabalho 2 de POO:

## 1. Funcionalidades implementadas

Está implementado no trabalho todas as funcionalidades exigidas no seu enunciado.

- Conjuntos extensíveis de blocos / Instanciação de blocos:

Há um pasta específica onde o usuário deve colocar os arquivos ".class" dos blocos disponíveis. A aplicação ao iniciar, lê desta pasta os arquivos e disponibiliza para o usuário a instanciação destes blocos, que aparecerão na sua área de trabalho.

- Movimentação de blocos / Conexão entre os blocos / Quebra da conexão / Remoção de blocos:

Estas funcionalidades constam no trabalho tais quais foram descritas no enunciado: O usuário poderá arrastar os blocos na tela, quandos estes forem colocados em posições vizinhas, eles serão conectados caso a interface deles permita. Caso um bloco seja removido ou movido da vizinhança de um outro bloco para fora desta, a conexão entre eles é desfeita.

- Atualização do estado:

Cada bloco deve fornecer uma método "Atualizar()" que é responsável por deixar o bloco num estado válido. Dependendo da implementação do bloco, este método pode ser chamado automaticamente quando este receber uma nova conexão. De qualquer forma, a aplicação chama este método de todos os blocos instanciados à cada vez que um de seus outros métodos é invocado.

- Persistência do ambiente

A aplicação é capaz de gravar num arquivo que tem um formato estilo *XML* a descrição do ambiente necessária para a recriação deste num momento futuro. Abaixo segue um exemplo deste arquivo:

```
<ConjuntoBlocos>
      <Bloco tipo="BlocoNumero" x="1" y="4">
            <atributo nome="Simbolo">2</atributo>
      </Bloco>
      <Bloco tipo="BlocoLetra" x="2" y="2">
            <atributo nome="Letra">N</atributo>
      </Bloco>
      <Bloco tipo="BlocoLetra" x="3" y="1">
            <atributo nome="Letra">V</atributo>
      </Bloco>
      <Bloco tipo="BlocoLetra" x="3" y="3">
            <atributo nome="Letra">B</atributo>
      </Bloco>
      <Bloco tipo="BlocoNumero" x="3" y="5">
            <atributo nome="Simbolo">5</atributo>
      </Bloco>
      <Bloco tipo="BlocoLetra" x="5" y="2">
            <atributo nome="Letra">A</atributo>
```

```
        </Bloco>
        <Bloco tipo="BlocoCor" x="9" y="2">
              <atributo nome="r">185</atributo>
              <atributo nome="g">102</atributo>
              <atributo nome="b">65</atributo>
        </Bloco>
   </ConjuntoBlocos>
```

O formato *XML* foi sugerido pelo professor e eu achei uma boa idéia usá-lo pois o *parser* já está pronto na biblioteca java, sua geração é simples, e para o caso em que se necessite mexer no arquivo fora da aplicação, esta seria uma tarefa simples, não exigindo nenhum editor melhor que o *Notepad*.

## 2. Especificação de como é realizada a troca de informações entre os blocos e a atualização de seus estados

Segue abaixo a listagem atual do arquivo Bloco.java e algumas observações:

```
package blocos.model;
import java.awt.Graphics;
import java.util.Map;

public interface Bloco
{
   public void ConectaBloco(Direcao d, Bloco b);
   public void Mostra(Graphics g);
   public void Atualizar();
   public void RemoveBloco(Direcao d);
   public Map<String, String> ObterAtributos();
   public  void  CarregarAtributos(Map<String,  String>  mapAtributos)
   throws ExcessaoAtributosInvalidos;

   public void DefinirTamanho(int TamanhoEmPixels);
   public void EventoClique(int x, int y);
   public void EventoMovimentoMouse(int x, int y);
   public void EventoRodaMouse(int x, int y, int valor);
   public void EventoTecla(String NomeTecla);
   public void EventoTimer();
}
```

Observações:

```
   public void ConectaBloco(Direcao d, Bloco b), e
   public void RemoveBloco(Direcao d):
```

Estes métodos devem ser chamados pela aplicação no momento da conexão ou desconexão de blocos. O argumento d pode assumir os valores do Enum Direcao definido no código, contento as constantes {cima, direita, baixo, esquerda}.

```
   public void Mostra(Graphics g):
```

Neste método o bloco deve se desenhar sobre o objeto g. Este objeto é do mesmo tipo do objeto g chamado pelo método show dos JComponents, sendo esta sua inspiração.

```
public Map<String, String> ObterAtributos(), e
public void CarregarAtributos(Map<String, String> mapAtributos):
```

Embora seja possível a aplicação investigar por reflexão quais são os atributos de cada bloco, achei melhor colocar na interface os métodos ObterAtributos e CarregarAtributos, onde através de um Map<String, String> pode-se fazer a comunicação entre a aplicação e o bloco sobre o valor dos seus atributos. Esta forma delega ao bloco a responsabilidade de lidar com a comunicação de seus atributos com a interface. Esta forma oferece algumas vantagens como simplificação da implementação e flexibilidade, uma vez que a quantidade de atributos do bloco pode variar de acordo com o seu estado.

## 3. Arquitetura

Neste trabalho estou procurei seguir o Padrão de Projeto "MVC" (Modelo-Visão-Controlador). Para isso coloquei as classes em três pacotes diferentes: blocos.model, blocos.view, e blocos.controller.

Pacote blocos.model:

Interface bloco (listada acima):

Todas as classes as quais se deseja instanciação como blocos na aplicação devem implementar esta interface (além de ter seu arquivo .class no diretório apropriado).

Enum Direcao (citada acima):

Enum com constantes "CIMA", "DIREITA", "BAIXO" e "ESQUERDA". Contém um método DirecaoOposta.

Classe CarregadorBlocos:

Reponsável pela investigação de quais blocos estão disponíveis para instanciação, e pelo disponibilização de um método para instanciar qualquer bloco disponível.

Pacote blocos.view:

Interface VisaoBlocos:

Define as mensagens as quais a camada de visão deve saber responder. Atualmente possui dois métodos: DesenhaBlocos e ObterGraphics. A segunda serve para a aplicação entregar para cada bloco um objeto Graphics que lhe dê poder de desenhar apenas na área que lhe foi designada.

Classe GridBlocos:

Classe que herda a implementação da classe javax.swing.JPanel. Responsável por desenhar a área de trabalho na tela.

Classe JanelaPrincipal:

Implementa a janela da aplicação que fornece os *menus* e a área de trabalho para o usuário atuar.

Pacote blocos.controller:

Classe GerenciadorDeBlocos:

Implementa o núcleo da aplicação: todas as funcionalidades de gerenciamento do ambiente onde os blocos estão. Implementa a parte de persistência, e gerencia quais blocos estão conectados a quais. Faz a interface entre a visão e o modelo.

Observações:

A parte de persistência do “BlocoPong” não pode ser implementada, o dicionário do “BlocoLetra” não pode ser implementado lendo de um arquivo e a janela de edição dos atributos do bloco não pode ser implementada, pois me enganei com a data do prazo de entrega do trabalho e acabei não reservando o tempo necessário.