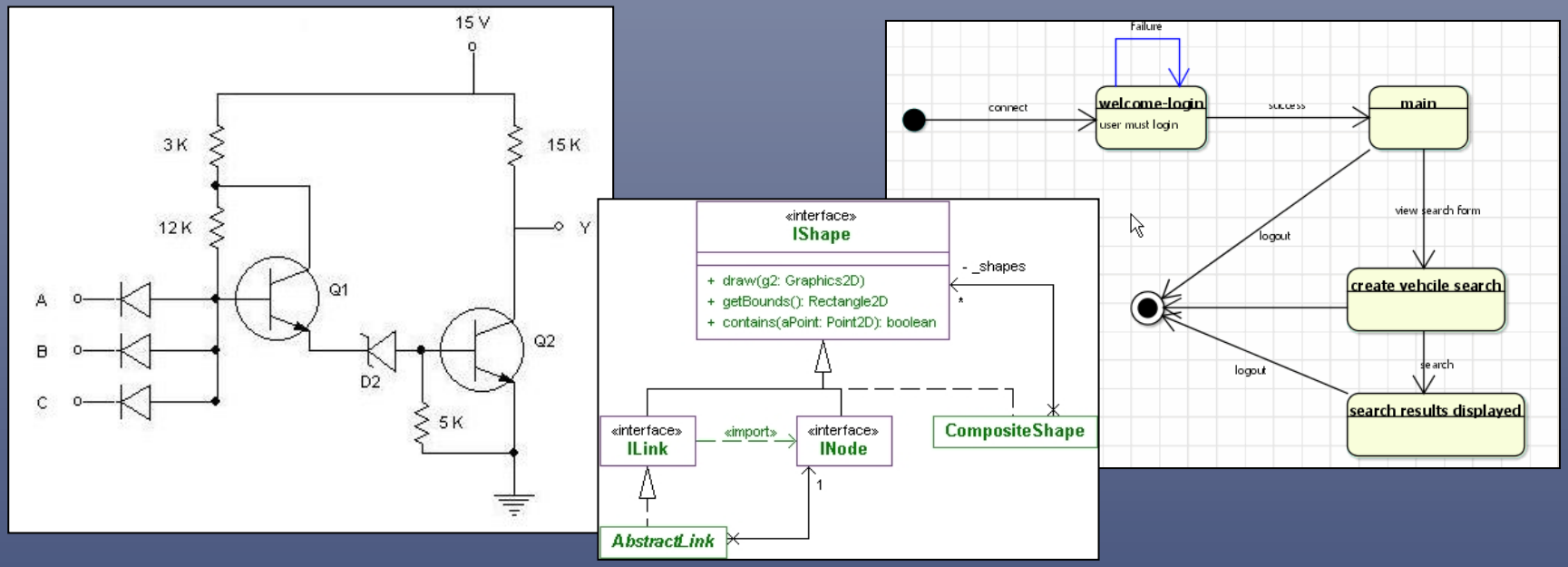
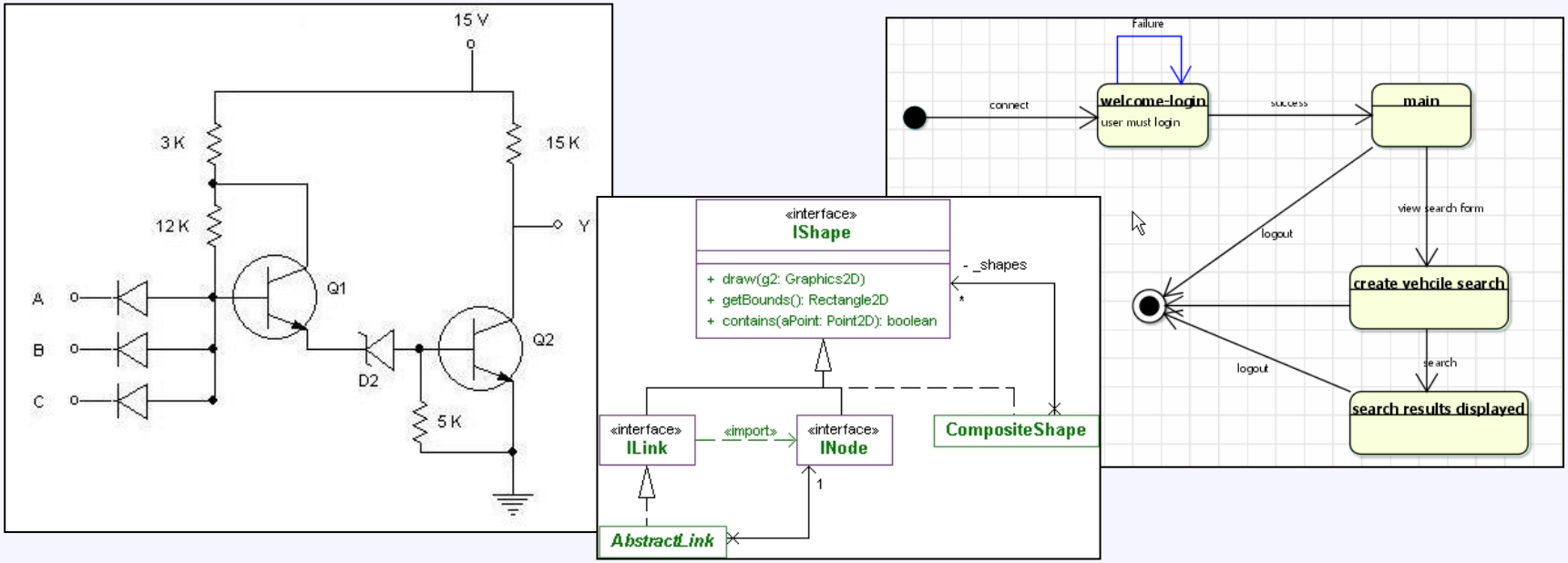
Framework gPad



gpad - Graphical Pad Framework

Domínio: **desenho interactivo de diagramas constituídos por nós e ligações**.

- No contexto de um diagrama de classes, os nós são rectângulos e as ligações são setas ou linhas com losangos.
- No contexto de um circuito electrónico, os nós serão transístores, diodos, resistências, etc, e as ligações são linhas simples.
- Muitos outros exemplos poderão ser considerados tais como, fluxogramas, circuitos lógicos, diagramas de estados, etc.



gpad… Objectivo

Objectivo da *framework gpad*:

- encapsular os aspectos comuns a todos os editores gráficos, em particular:
  - a interface com o utilizador,
  - tratamento de comandos e *inputs* do utilizador (rato ou teclado);

- providenciar uma forma dos tipos específicos de determinados diagramas expressarem as intenções que estão por de trás dos serviços da *framework*.

Divisão de Responsabilidades

Ao desenhar-se uma *framework* é necessário dividir as responsabilidades entre a *framework* e o que é especifico de uma determinada aplicação.

Exemplos de **responsabilidades de uma aplicação concreta**:

- Desenhar a forma de um nó (exemplos: transístor, tipos, etc);
- Detectar se um ponto está sobre um nó, ou uma ligação, está dependente da forma mais, ou menos complexa da figura (por exemplo, um ponto resultante do "click" do rato);
- Retornar os limites de espaço de uma figura.

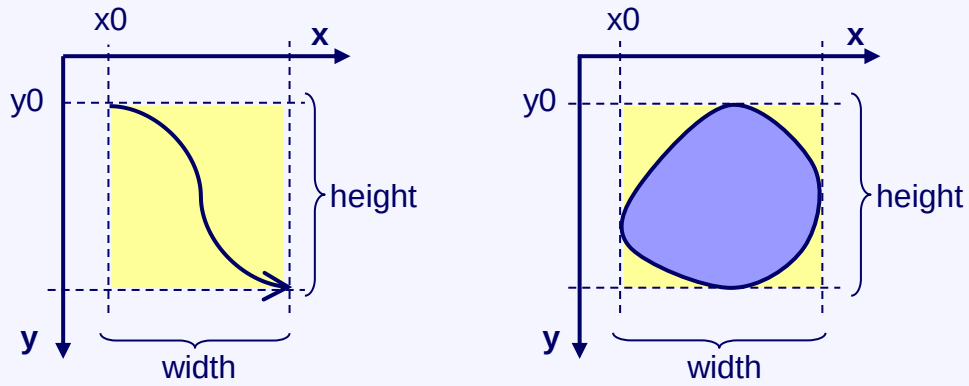
Exemplos de **responsabilidades da *framework gpad***:

- Desenhar a *toolbar*;
- Reajustar as dimensões da UI ao espaço ocupado por uma figura;
- Determinar a acção resultante dos *inputs* do utilizador, tais como criação de nós e ligações sobre o componente de desenho.

Organização do *gpad*

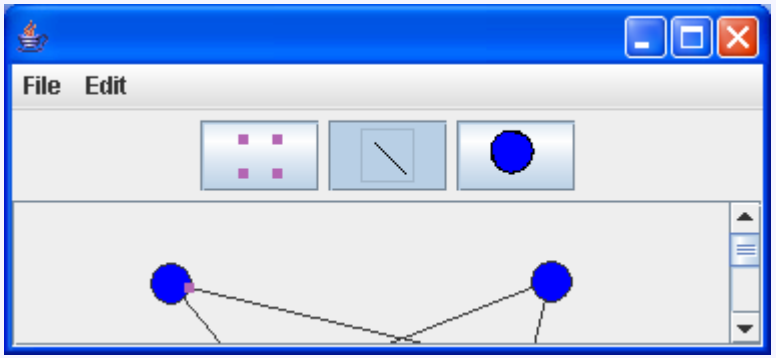
Modelo

Especificação das entidades manipuladas pela *framework*: **nós** e **ligações**.

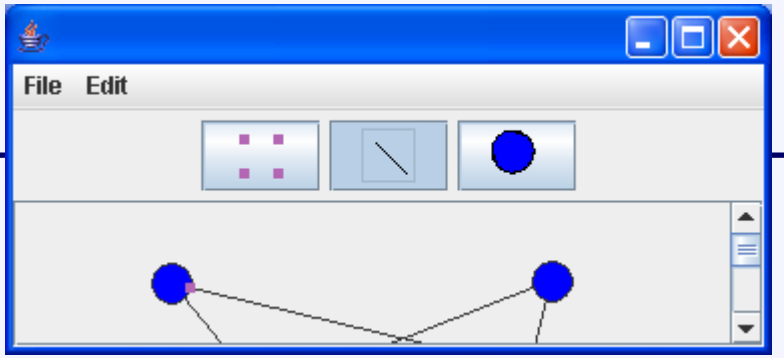


UI – *User Interface*

Interacção com o utilizador.



Organização do *gpad… UI*



toolbar no topo com um botão para cada tipo de nó e cada tipo de ligação:

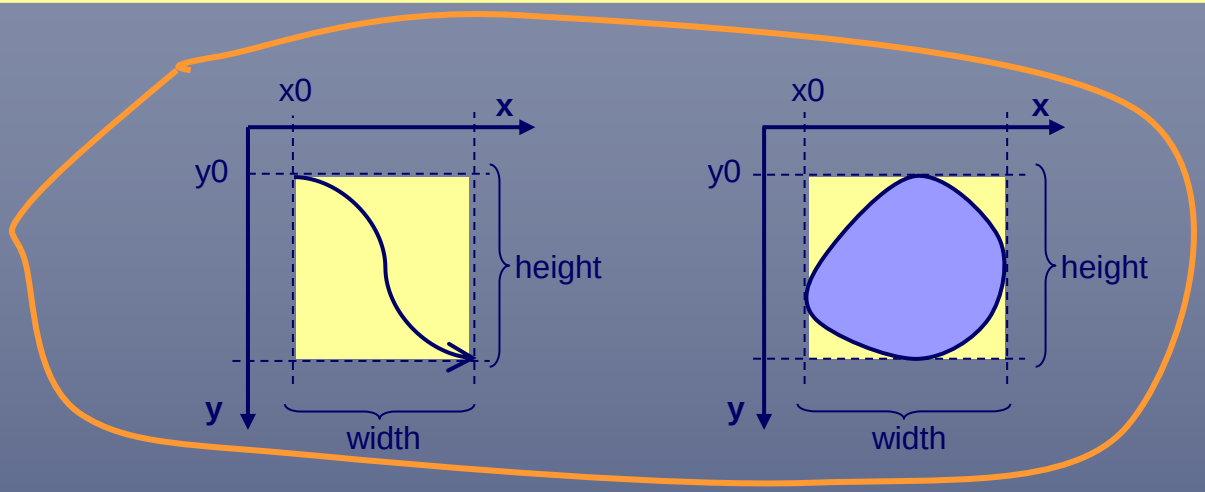
- o botão mais à esquerda é o ***grabber***, que serve para seleccionar os nós e ligações;
- num determinado momento só estará um único botão activo.

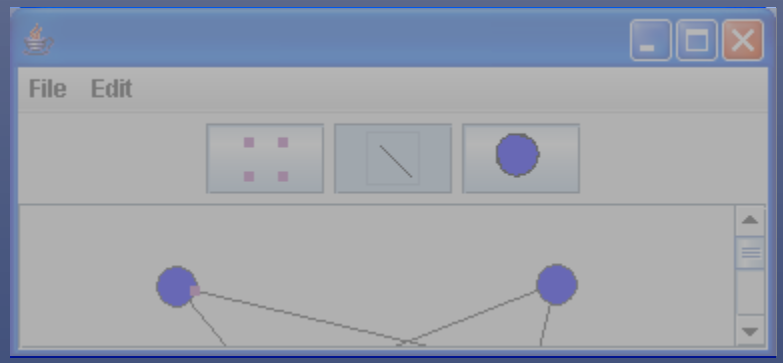
Menu de opções para carregar e gravar o diagrama, apagar o nó ou ligação seleccionada e editar propriedades.

Componente de desenho ocupa o centro do ecrã:

- Mantém o desenho de todas as figuras e linhas adicionadas;
- No caso de um elemento estar seleccionado desenha as marcas de selecção sobre esse elemento;
- o rato é usado para desenhar;
- o utilizador pode "clicar" sobre um nó, uma ligação ou no espaço vazio;
- o utilizador pode usar o rato para ligar nós ou deslocar um nó para uma nova posição;
- as acções do rato dependem do local onde o utilizador "clicou" e do botão que estiver seleccionado na ***toolbar***.

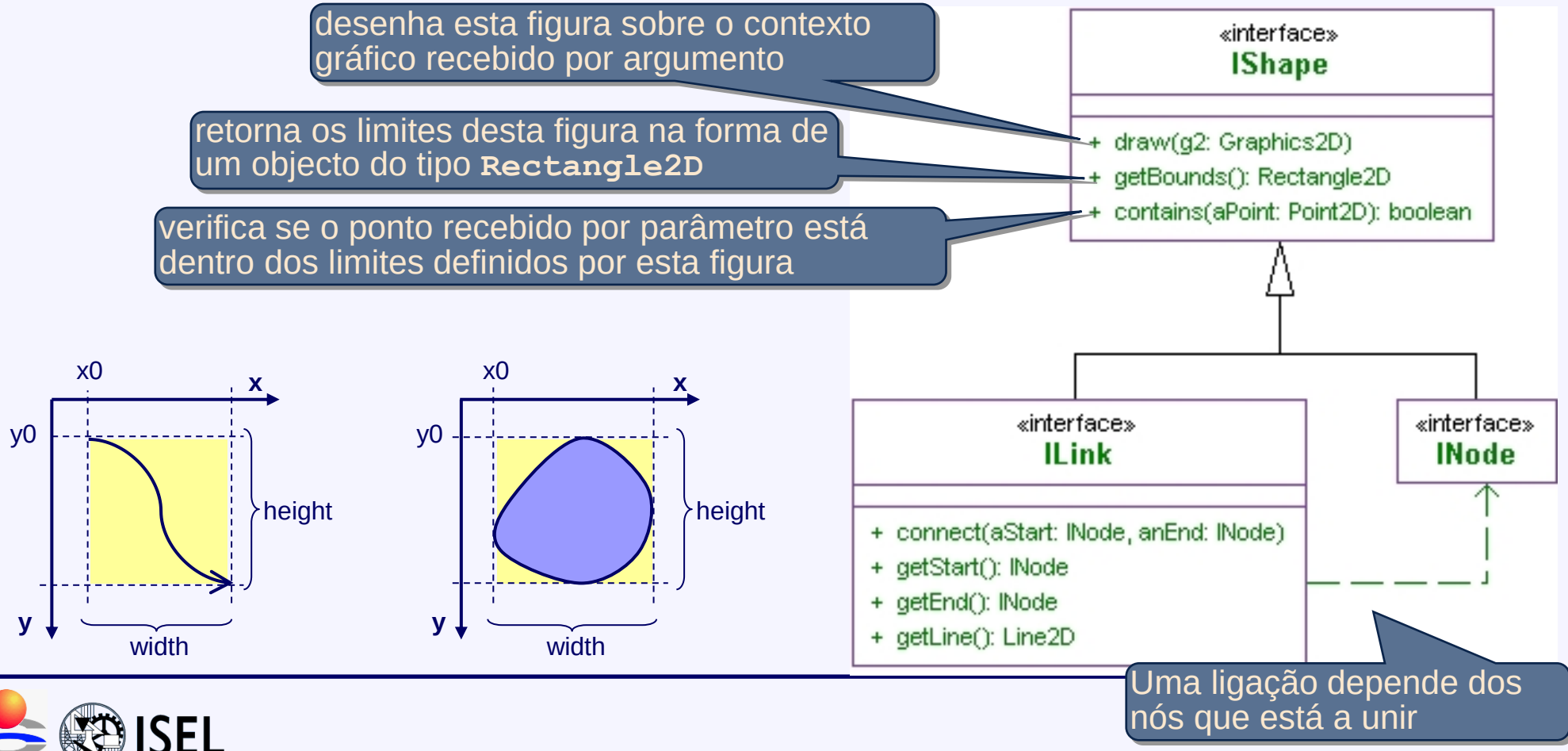
Framework gPad... Modelo





Modelo *gpad*

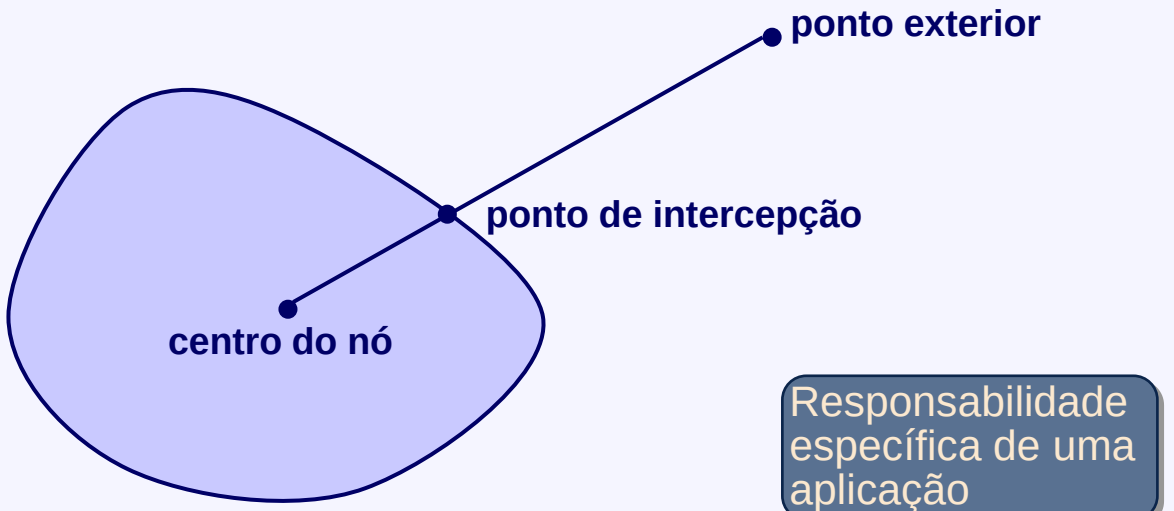
- Qualquer figura suportada pela *framework gpad* é subtipo de **`IShape`**.
  `IShape` especifica tudo aquilo que é comum aos **nós** e **ligações**.
- Por sua vez, os nós são subtipos de **`INode`** e as ligações de **`ILink`**.



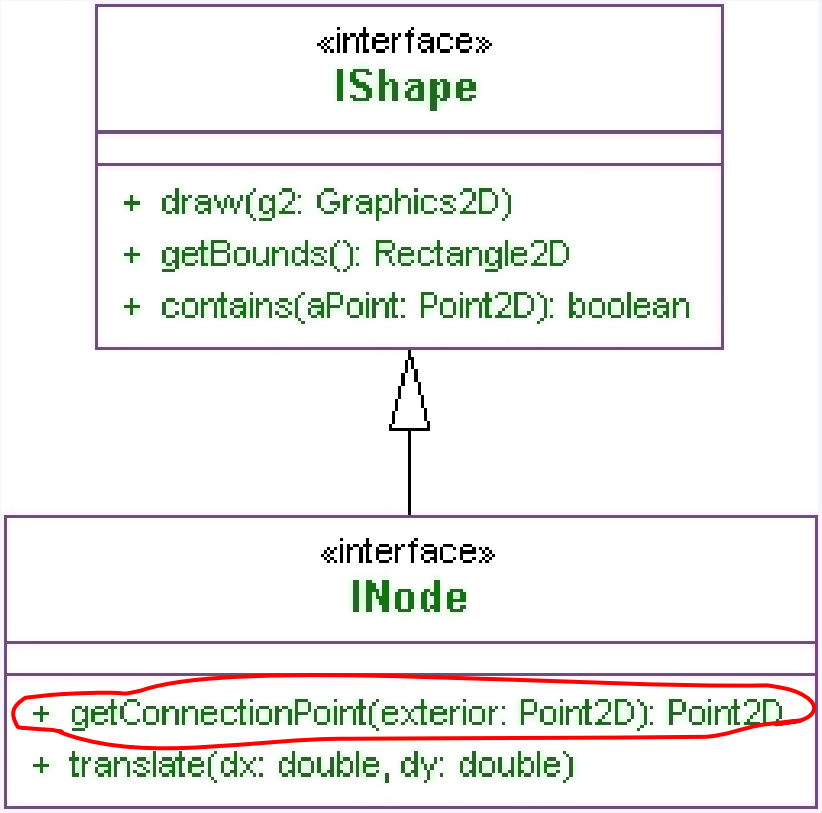
Modelo *gpad*... `INode`

- Para desenhar a ligação entre dois nós é necessário conhecer os **pontos de intercepção** na fronteira de cada nó.
- O determinação do ponto de intercepção depende da forma da figura e deverá ser implementado por cada nó concreto através do método:

  `getConnectionPoint(Point2D exterior)`

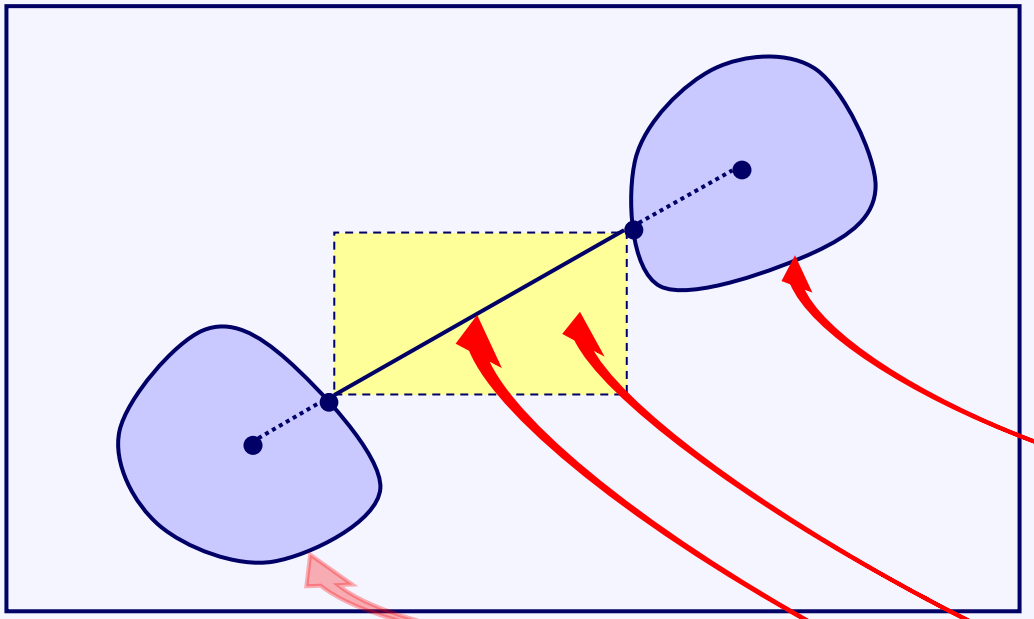


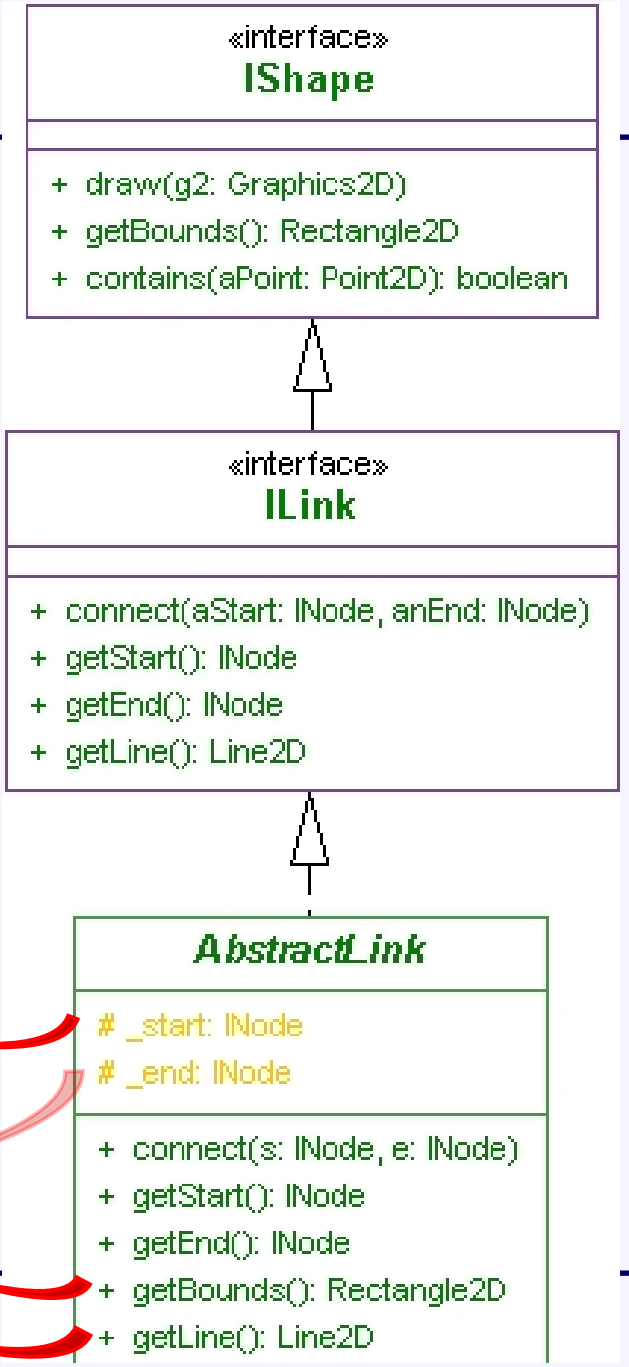
Responsabilidade específica de uma aplicação



Modelo *gpad*... `AbstractLink`

- `AbstractLink` implementa a funcionalidade comum a todas as ligações.
- Para cada ligação concreta resta implementar os métodos `draw` e `contains`.

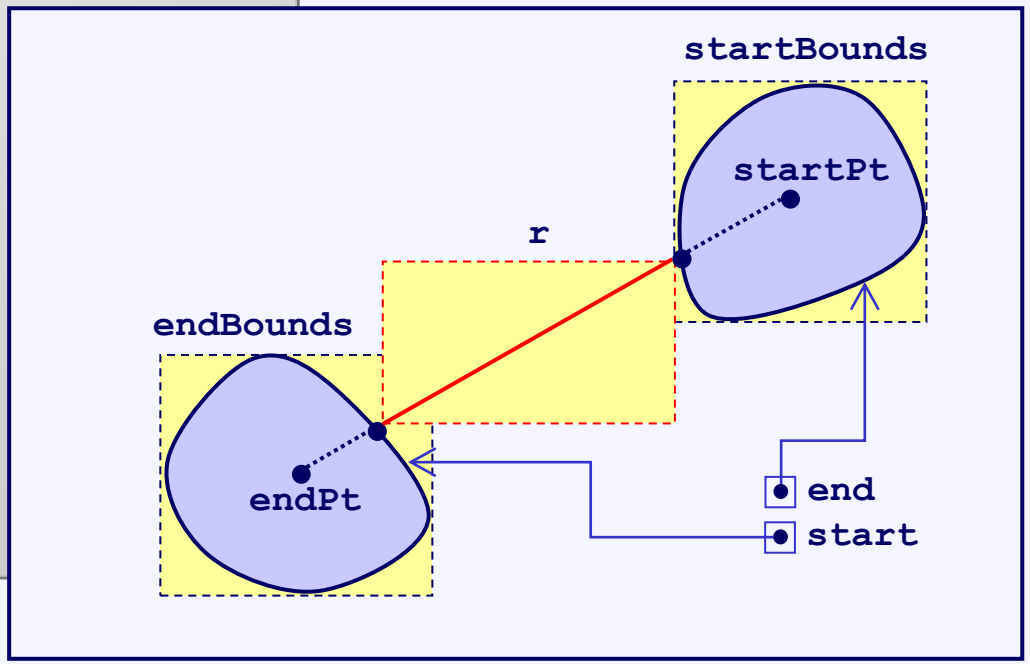




ISEL

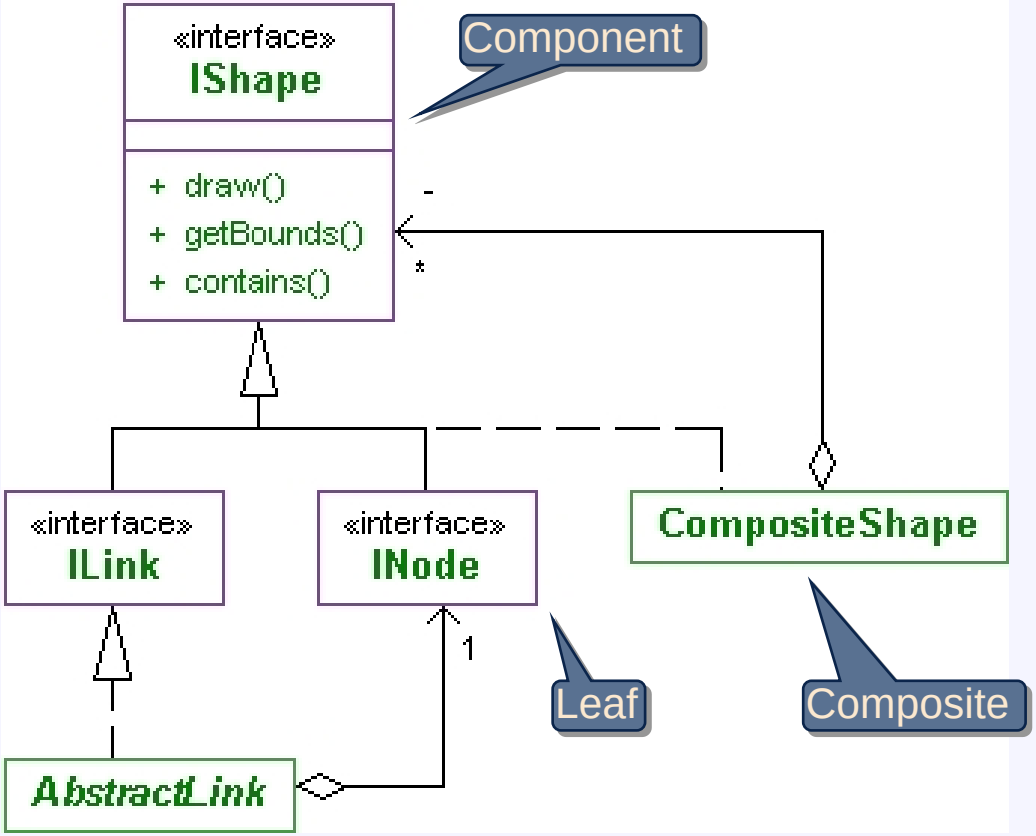
Modelo *gpad*... `AbstractLink`...

```
public Line2D getLine(){
  Rectangle2D startBounds = start.getBounds();
  Rectangle2D endBounds = end.getBounds();
  Point2D startPt = new Point2D.Double(
    startBounds.getCenterX(), startBounds.getCenterY());
  Point2D endPt = new Point2D.Double(
    endBounds.getCenterX(), endBounds.getCenterY());
  return new Line2D.Double(
    start.getConnectionPoint(endPt),
    end.getConnectionPoint(startPt));
}
public Rectangle2D getBounds(){
  Line2D conn = getLine();
  Rectangle2D r = new Rectangle2D.Double();
  r.setFrameFromDiagonal(
    conn.getX1(), conn.getY1(),
    conn.getX2(), conn.getY2());
  return r;
}
```



Modelo *gpad*... `CompositeShape`

`CompositeShape` define uma forma composta constituída por um conjunto de instâncias subtipo de `IShape` (aplicação do padrão ***composite***).



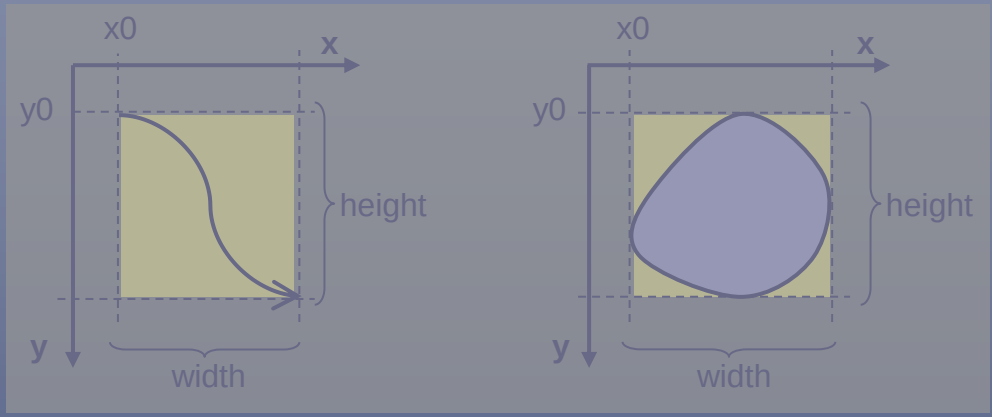
Característica:

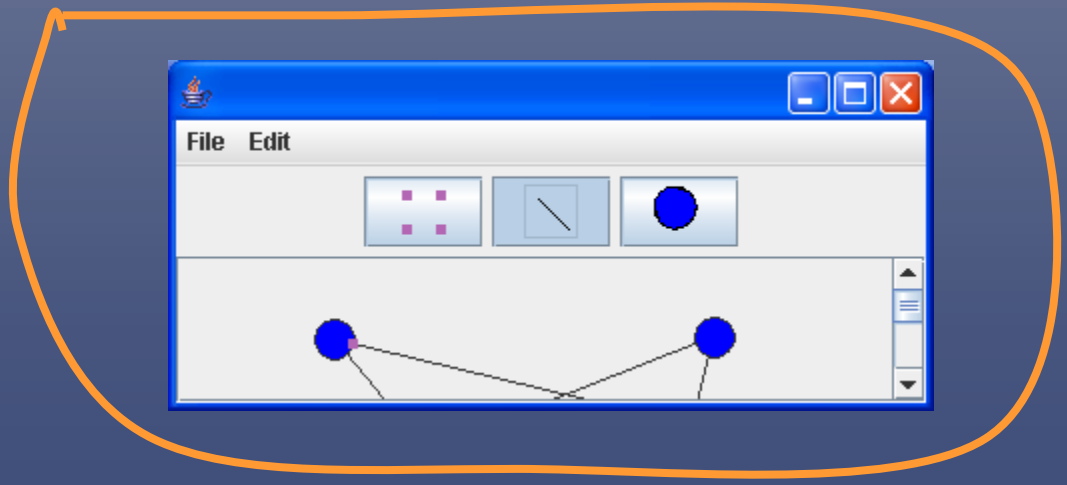
➔ Especifica como devem ser combinados diferentes objectos num único objecto, de modo a que **o objecto composto tenha o mesmo comportamento de cada uma das suas partes individuais**.

Modelo *gpad*... `CompositeShape` ...

```
/**
 * Draws the shape
 * @param g2 the graphics context
 */
public void draw(Graphics2D g2){
  for (IShape n : _shapes)
    n.draw(g2);
}
/**
 * Gets the smallest rectangle enclosing the shape
 * @param g2 the graphics context
 * @return the bounding rectangle
 */
public Rectangle2D getBounds(){
  Rectangle2D r = null;
  for (IShape n : _shapes){
    Rectangle2D b = n.getBounds();
  if (r == null) r = b;
    else r.add(b);
  }
  return r == null ? new Rectangle2D.Float() : r;
}
```

Framework gPad... UI

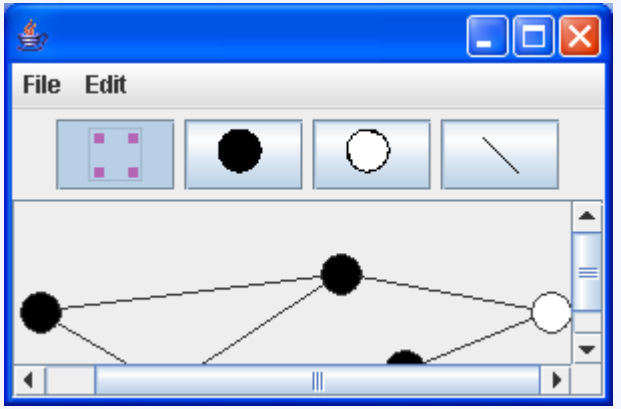




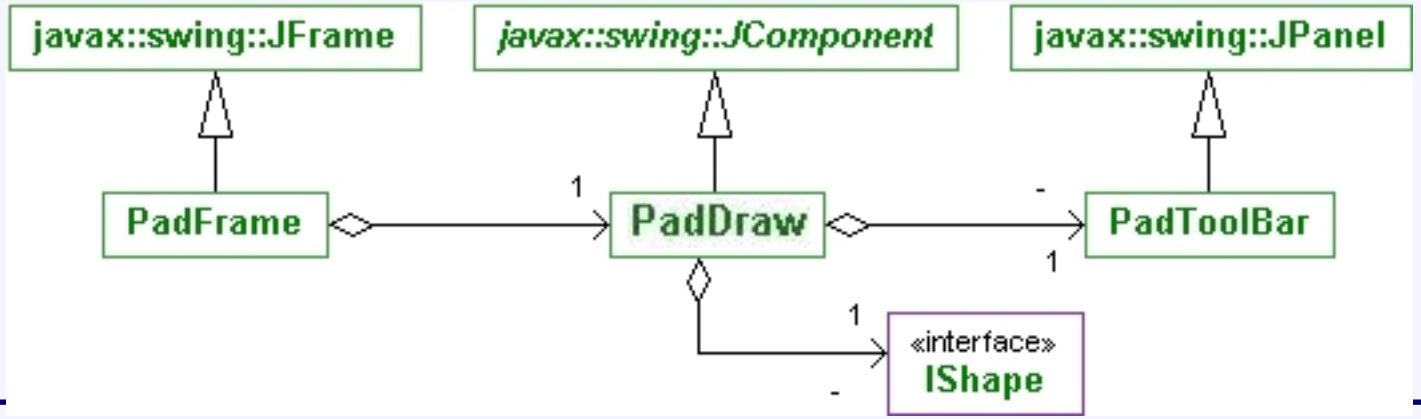
UI gpad

A responsabilidade da *user interface* da *framework gpad* está dividida pelas classes:

- **`PadFrame`**: *frame* que gere a *toolbar*, os menus de opções e o componente de desenho (`PadDraw`);
- **`PadDraw`**:
  - desenha a forma definida por uma instância de `IShape`;
  - faz o tratamento das operações do rato em conjunção com os comandos seleccionados na *tollbar*.
- **`PadToolbar`**: painel de botões de escolha única, correspondentes aos vários tipos de nós e ligações disponíveis.

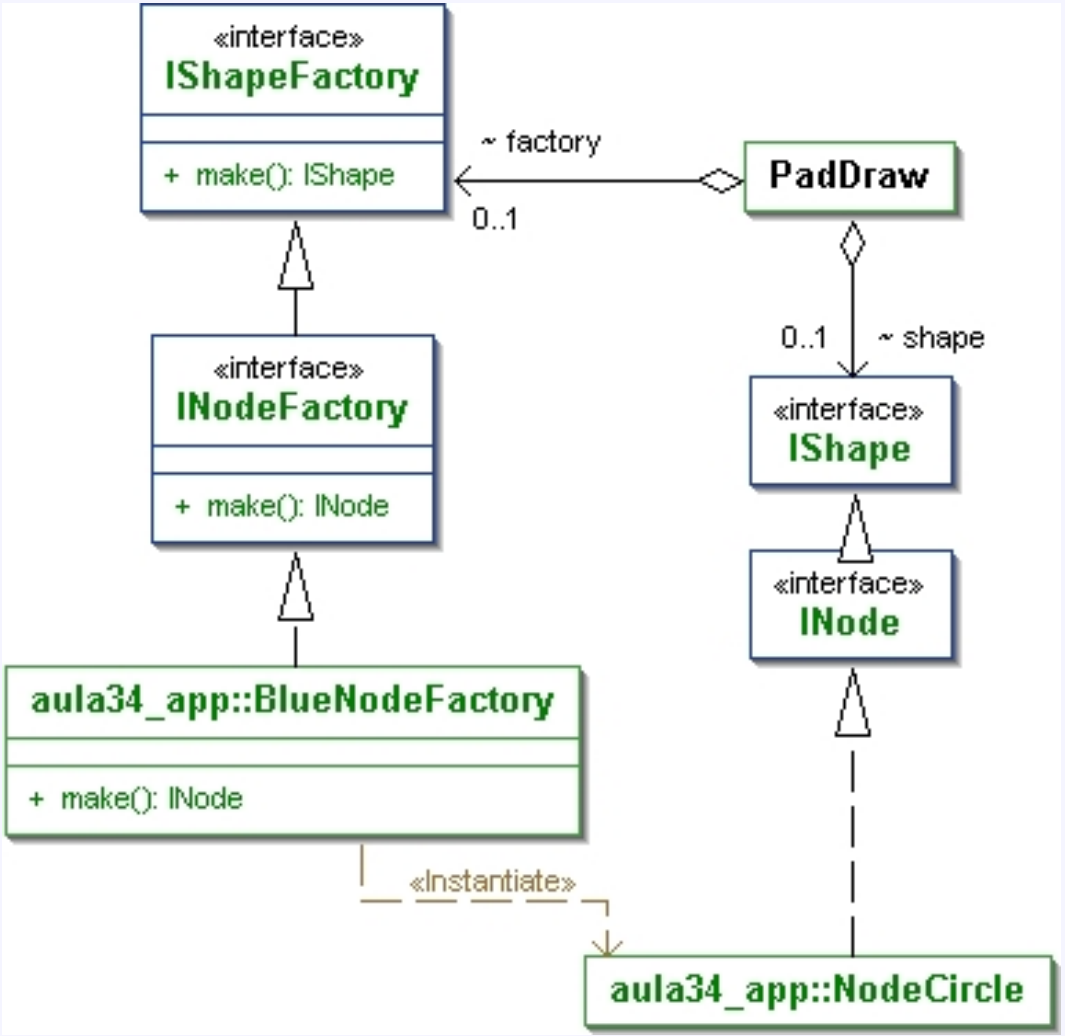


1ª Abordagem:

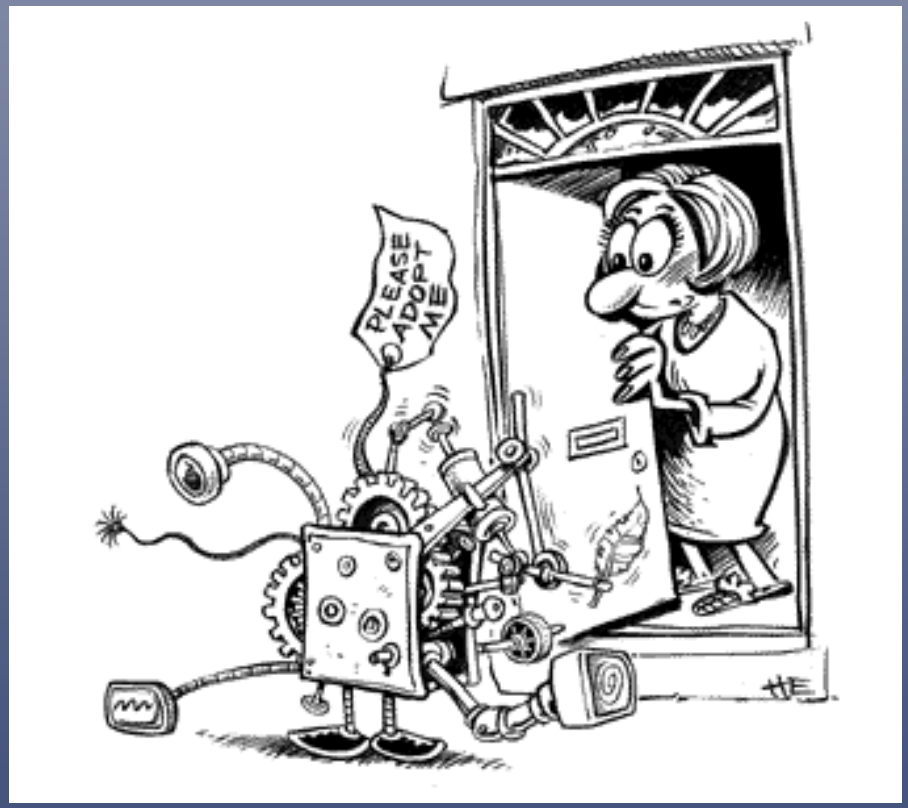


UI gpad… Abstract Factory

- As funcionalidades de `PadDraw` estão dependentes da capacidade de saber criar instâncias de subclasses de `INode` e `ILink` (desconhecidas em *gpad*).
- Essa funcionalidade é definida nas interfaces `INodeFactory` e `ILinkFactory` que especificam o comportamento de classes capazes de produzir objectos compatíveis com `INode` e `ILink`, respectivamente.
- Cada botão tem associado uma instância de uma *factory*.

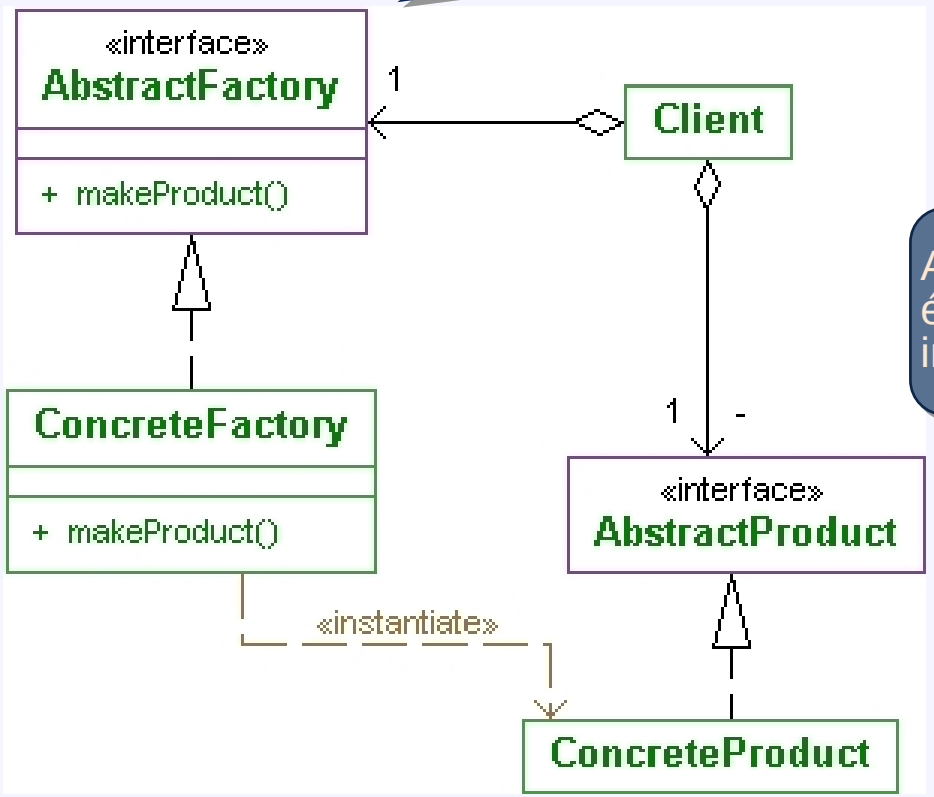


Padrão *Abstract Factory*



Padrão *Abstract Factory*

O cliente deve ser independente da forma como os seus componentes ou produtos são criados.



Apenas a interface dos produtos é exposta, enquanto que a implementação não é relevante.

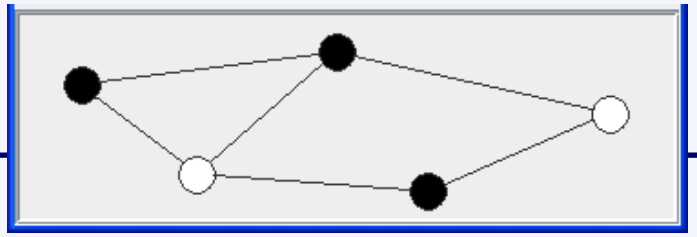
Padrão *Abstract Factory* - Participantes

Nome do Participante	Descrição
`Product` (`INode e ILink`)	Interface que define os objectos a criar pela *Factory.*
`ConcreteProduct` (`BlackNode, LineEdge, ...`)	Implementa a interface `Product`.
`AbstractFactory` (`INodeFactory, ILinkFactory`)	Define o *factory method* que retorna um objecto do tipo produto.
`ConcreteFactory` (`BlackNodeFactory, LineEdgeFactory,...`)	Redefine o *factory method* e retorna uma instância de `ConcreteProduct`.

Padrão *Abstract Factory*

Característica	Descrição
Nome	*Abstract Factory*
Nome alternativo	*Kit*
Categoria	Criação – Objecto
Objectivo	Disponibilizar uma interface para a criação de uma família de objectos relacionados ou dependentes sem conhecer as suas classes concretas.
Aplicabilidade	• um sistema deve ser independente da forma como os seus componentes ou produtos são criados. • um sistema deve ser configurável com um de múltiplos produtos da mesma família. • apenas a interface dos produtos é exposta, enquanto que a implementação não é relevante.

UI gpad… `PadDraw`… Interacção



Tratamento das acções do rato

Quando o botão seleccionado na *toolbar* corresponder a um:

- ***grabber*** – o "*click*" sobre um nó ou uma ligação torna-o seleccionado;
- ***grabber*** – iniciar uma operação de arrastar dentro de um nó fá-lo deslocar-se assim como as ligações que estão associadas a si;
- **nó** - o "*click*" sobre um espaço vazio insere um novo nó no componente de desenho. O tipo de nó criado depende do botão seleccionado na *toolbar*;
- **ligação** – iniciar uma operação de arrastar dentro de um nó e arrastar o cursor até outro nó, insere uma nova ligação. O seu tipo corresponde ao tipo de ligação seleccionado na *toolbar*.

Característica:

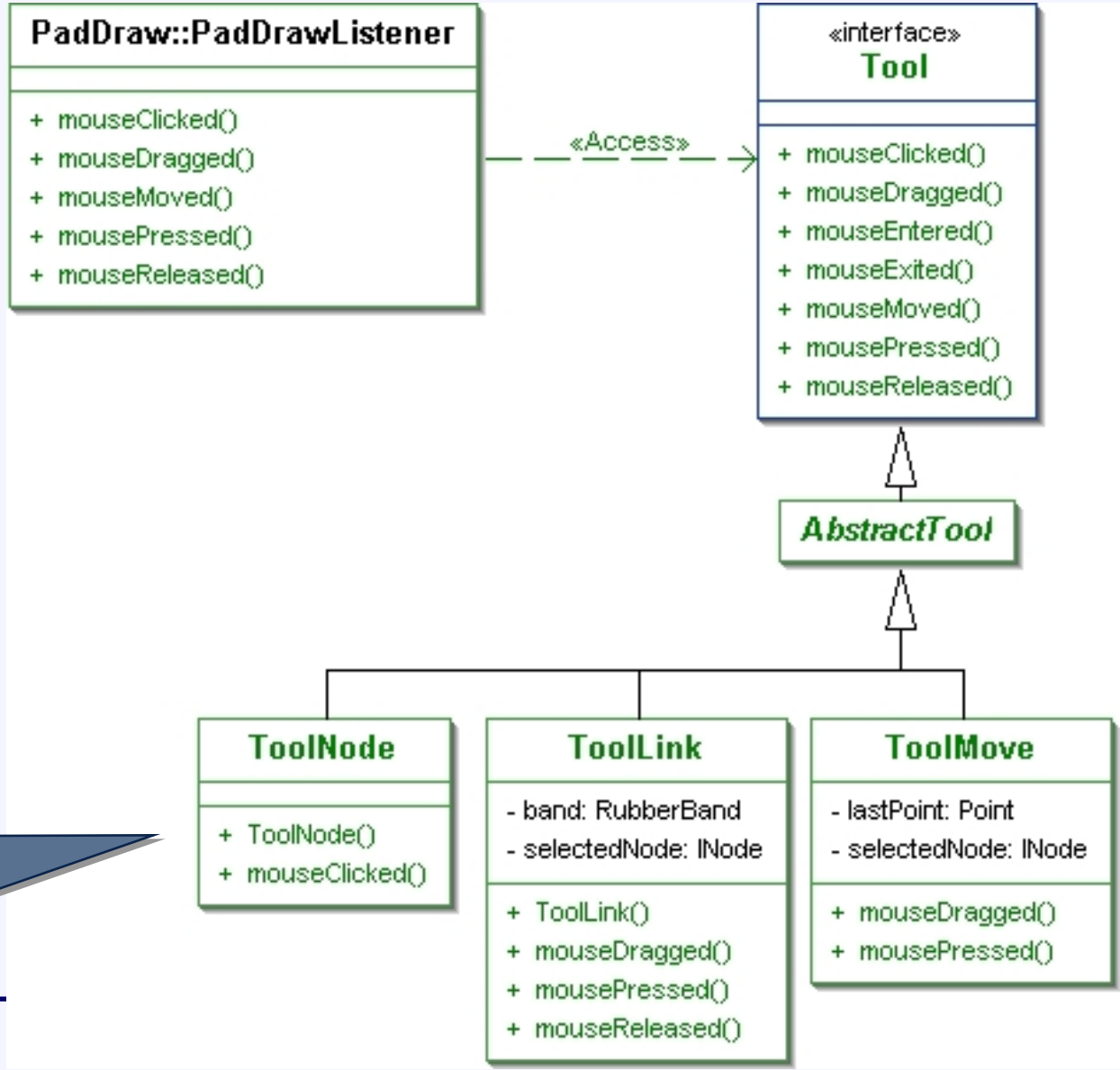
- A mesma acção do rato tem comportamentos diferentes dependendo do contexto de utilização (botão seleccionado, alvo do rato, etc).

➔ O comportamento de `PadDraw` depende do seu **estado**, que é actualizado em tempo de execução mediante a combinação de diversas condições.

UI gpad… `PadDraw…` `Tool`

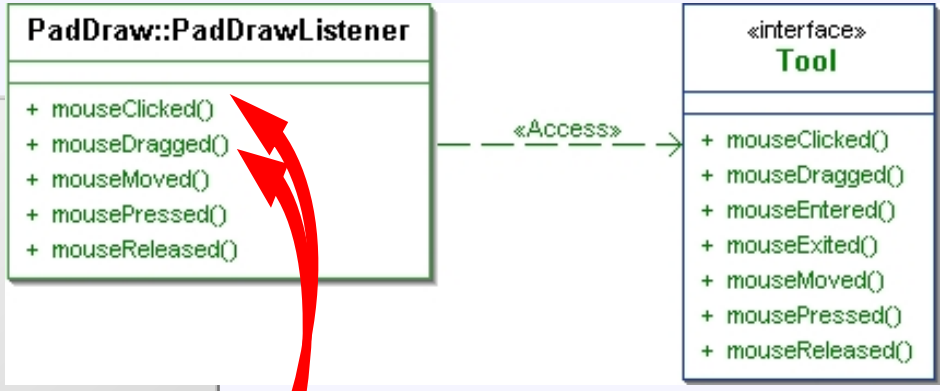
- O tipo abstracto `Tool` especifica o comportamento de `PadDraw`.
- As subclasses de `Tool` implementam o comportamento específico de `PadDraw` para determinadas situações.
- `PadDrawListener` delega os seus pedidos numa instância de uma subclasse de `Tool`.

Cada implementação concreta de `Tool` reflecte as acções do utilizador de uma determinada forma sobre a instância de `IShape` que está a ser actualizada.



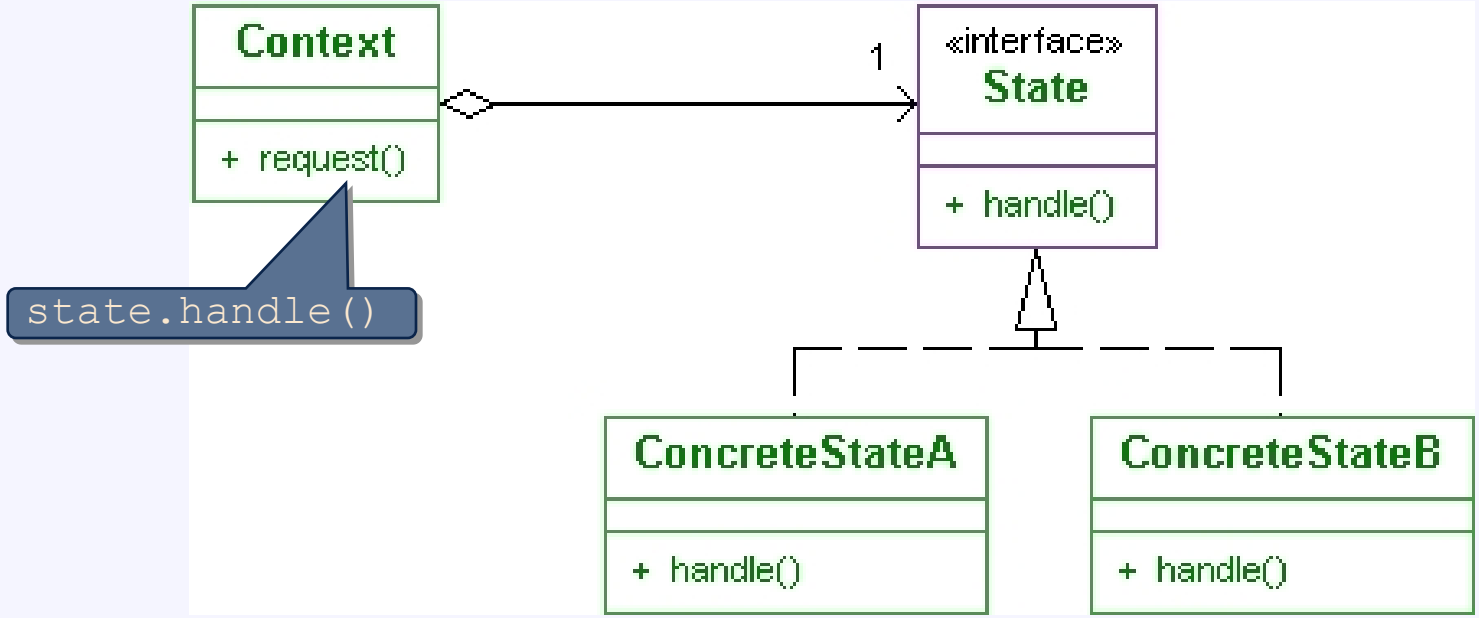
UI gpad... PadDraw... Tool...

```
public class PadDraw extends JComponent{
   public PadDraw(PadToolBar aToolBar){
      ...
      PadDrawListener h = new PadDrawListener();
      addMouseListener(h);
      addMouseMotionListener(h);
   }
   /**
    * MouseListener and MouseMotionListener implementation
    */
   private class PadDrawListener
     extends MouseAdapter
     implements MouseMotionListener{
      public void mouseClicked(MouseEvent e){
          Tool t = tBar.getSelectedTool();
          t.mouseClicked(e, shape);
          repaint();
      }
      public void mouseDragged(MouseEvent e){
          Tool t = tBar.getSelectedTool();
          t.mouseDragged(e, shape);
          repaint();
      }
    ...
```



Padrão *State*

- O desenho da solução aplicada a `PadDraw` e `Tool` segue o padrão State, que encapsula o comportamento de cada ferramenta numa classe separada e desacopla simultaneamente as ferramentas da classe `PadDraw`.



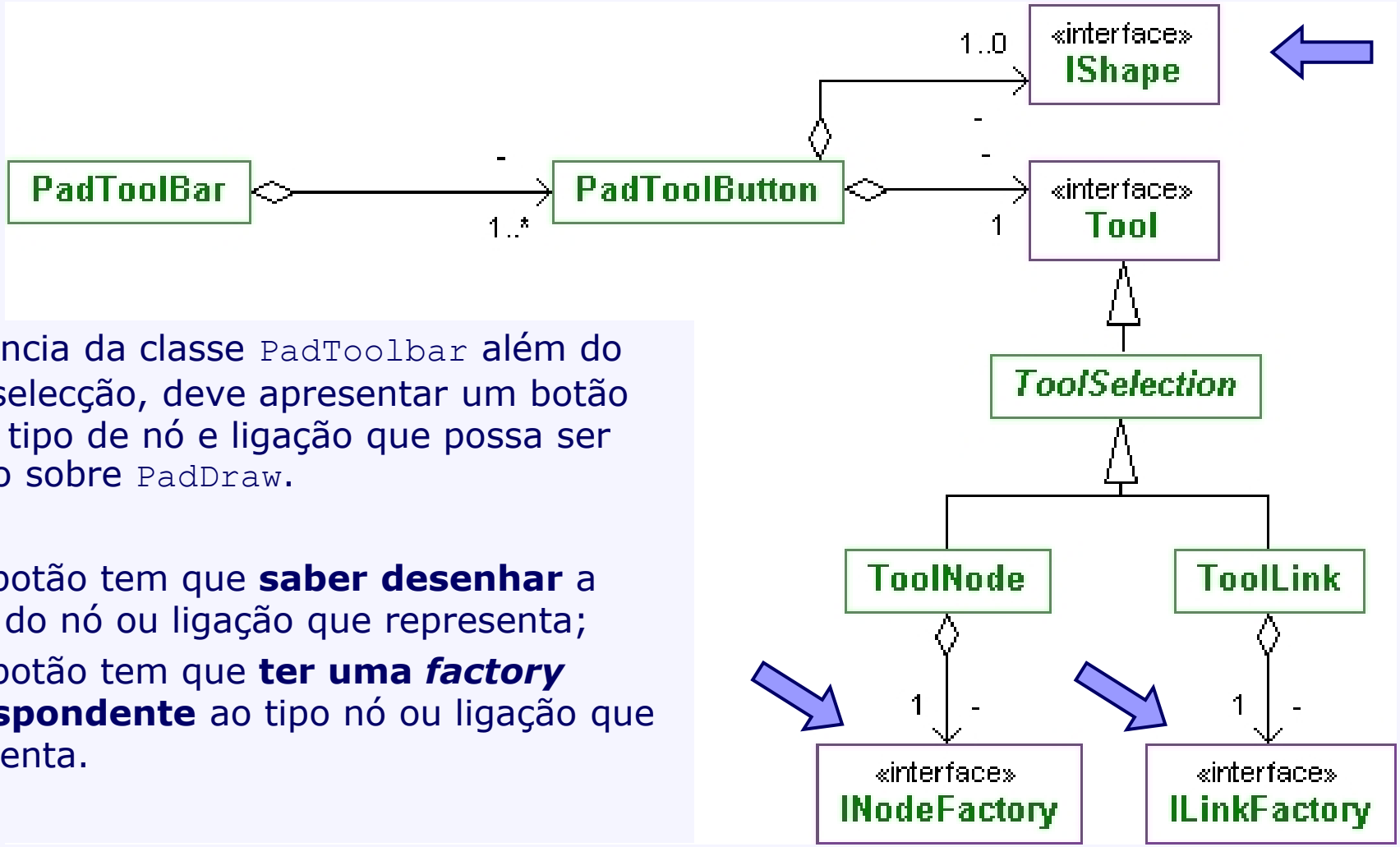
Padrão *State* - Participantes

Nome do Participante	Descrição
`Context` (`PadDrawListener`)	Mantém uma instância de `ConcreteState` que define o seu estado corrente (ex: Tool t = tBar.getSelectedTool()).
`State` (`Tool`)	Interface que especifica o comportamento encapsulado num determinado estado particular de `Context`.
`ConcreteState` (`ToolMove`, `ToolNode`, `ToolLink`)	Cada subclasse implementa o comportamento associado a um determinado estado de `Context`.

Padrão *State*

Característica	Descrição
Nome	*State*
Categoria	Comportamento – Objecto
Objectivo	Ajudar um objecto a alterar a implementação do seu comportamento em tempo de execução, quando o seu estado interno se altera.
Aplicabilidade	• quando a implementação do comportamento de um objecto depende do seu estado e esta deve alterar-se em tempo de execução. • quando os métodos têm múltiplos excertos de instruções que dependem do estado do objecto (por exemplo uma construção baseada em `switch` com longos excertos de código em cada caso).

UI gpad... `PadTollbar`



Uma instância da classe `PadToolbar` além do botão de selecção, deve apresentar um botão para cada tipo de nó e ligação que possa ser desenhado sobre `PadDraw`.

Desafios:

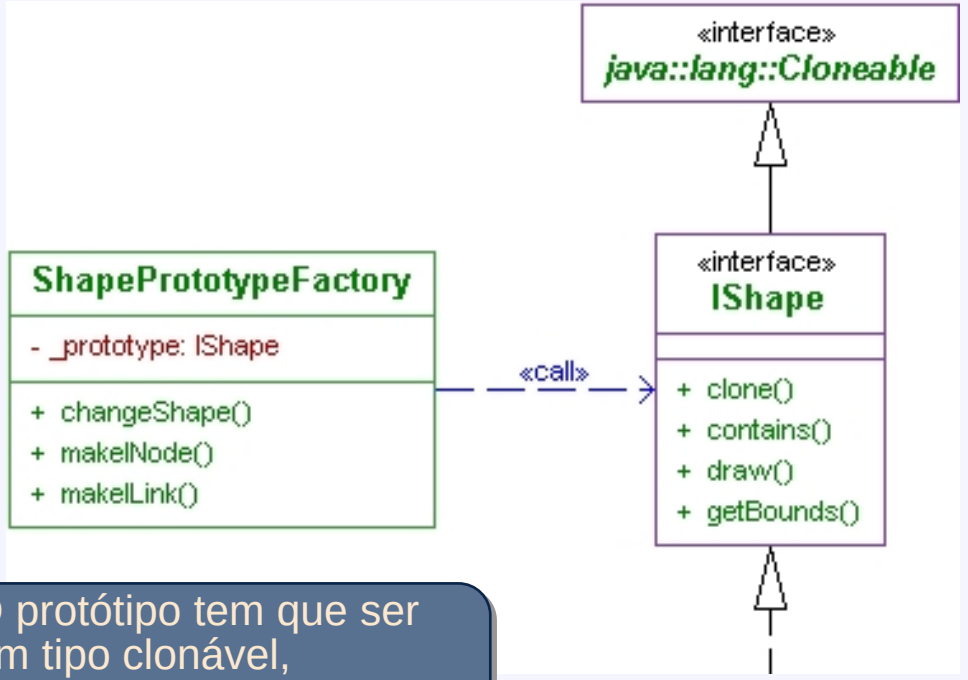
- Cada botão tem que **saber desenhar** a forma do nó ou ligação que representa;
- Cada botão tem que **ter uma *factory* correspondente** ao tipo nó ou ligação que representa.

UI gpad... `PadTollbar`...

Ter uma *factory* que cria instâncias de subclasse de `IShape`, clonando o objecto `IShape` que estiver associado ao botão seleccionado.

➔ Só é necessária uma instância dessa *factory*;

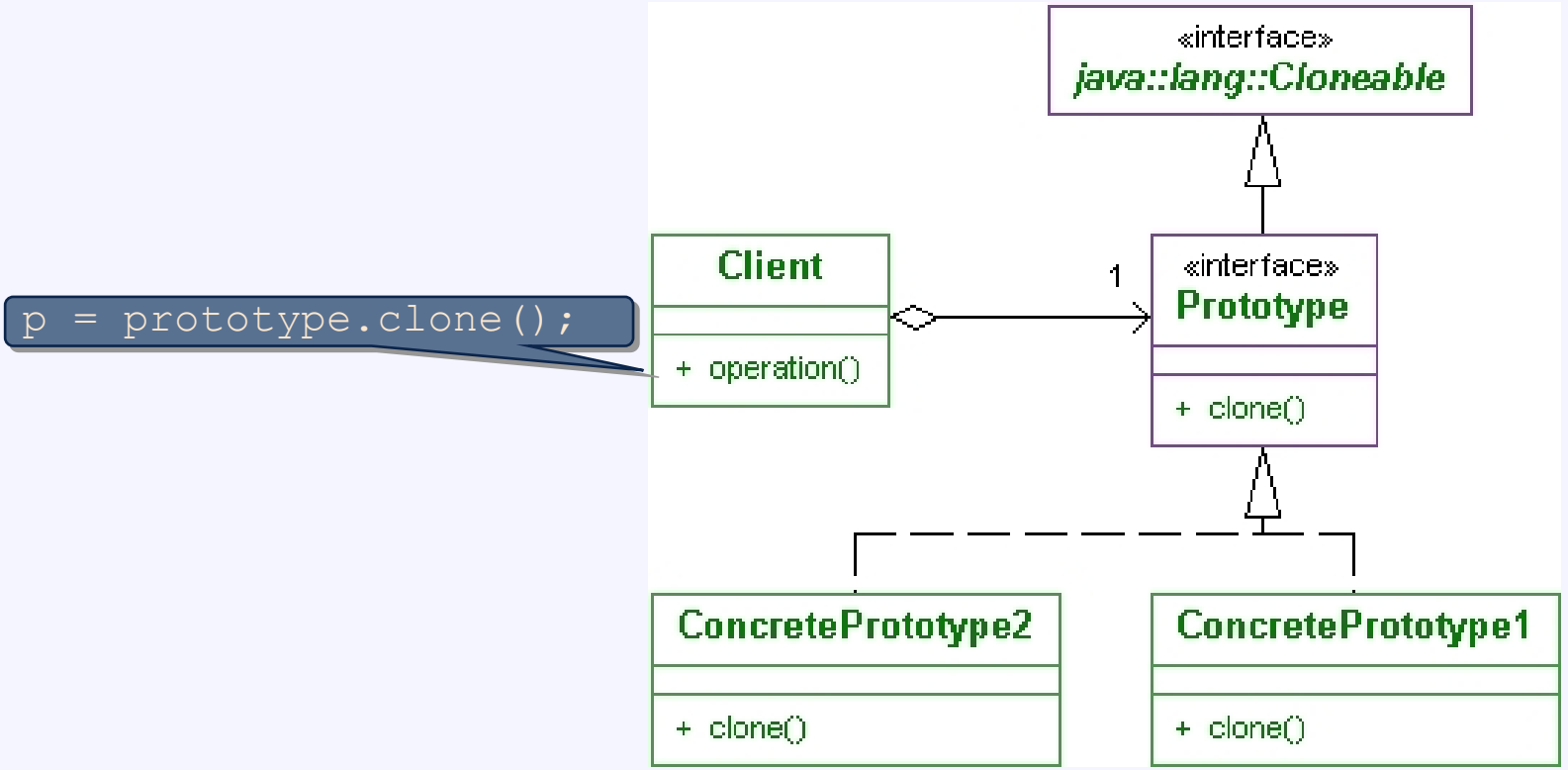
```
public class ShapePrototypeFactory
 implements INodeFactory, ILinkFactory{
  public void changeShape(IShape shape){
     _prototype = shape;
  }
  public INode makeINode(){
     if(_prototype instanceof INode)
       return (INode) _prototype.clone();
     else
       return null;
  }
  public ILink makeILink() {
     if(_prototype instanceof ILink)
       return (ILink) _prototype.clone();
     else
       return null;
     }
  private IShape _prototype;
}
```



O protótipo tem que ser um tipo clonável, implementar a interface `ICloneable`.

Padrão *Prototype*

- O tipo de objectos instanciado pode ser alterado em tempo de execução actualizando o respectivo protótipo.
- Evitar construir um hierarquia de classes de *factories* que estão em paralelo com uma hierarquia de produtos.



Padrão *Prototype* - Participantes

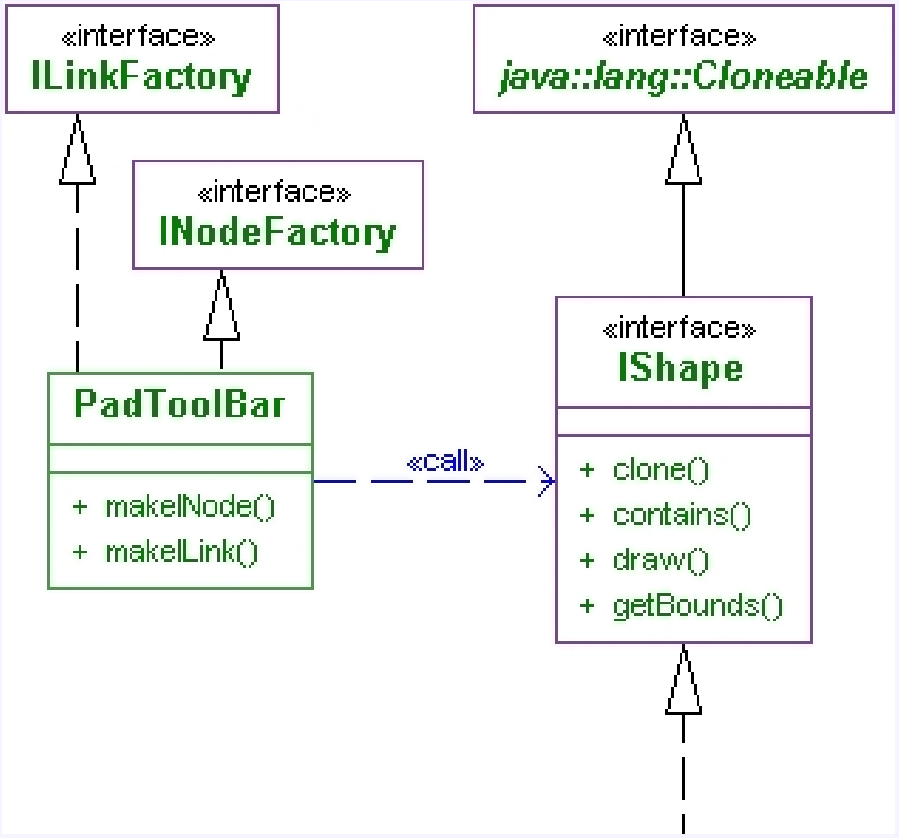
Nome do Participante	Descrição
`Prototype (INode, ILink)`	Define a interface dos objectos que serão criados, implementa a interface `Cloneable` e define um método público `clone`().
`ConcretePrototype (BlackNode, LineEdge, etc)`	Implementação da interface definida em `Prototype` e do método `clone`().
`Client (PadToolBar)`	Cria novas instâncias clonando os protótipos.

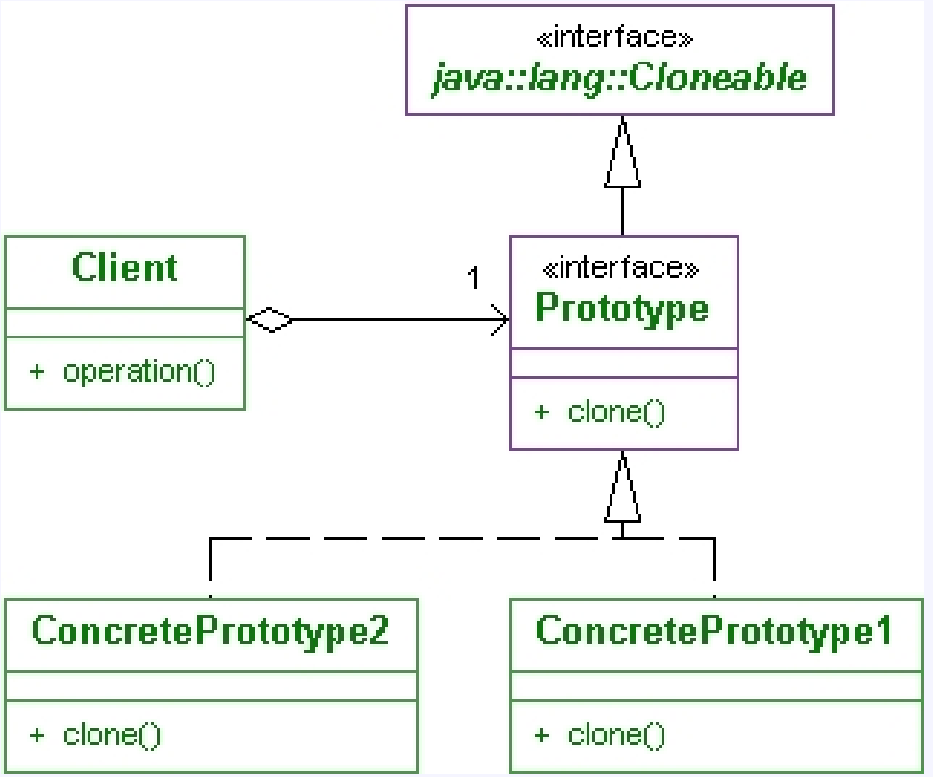
Padrão *Prototype*

Característica	Descrição
Nome	*Cloneable*
Categoria	Criação – Objectos
Objectivo	Especifica o tipo de objectos a criar com base numa instância protótipo e crias as novas instâncias clonando esse protótipo.
Aplicabilidade	• Quando o sistema deve ser independente da forma como os seus componentes ou produtos são criados. • Quando as classes a instanciar são especificadas em tempo de execução. • Evitar construir um hierarquia de classes de *factories* que estão em paralelo com uma hierarquia de produtos.

Modelo *gpad*... `IShape` ... Prototype

- Através do padrão `Prototype` os objectos são criados clonando os seus protótipos.
- O tipo de objectos criado pode ser alterado em tempo de execução actualizando o respectivo protótipo.





Aplicações sobre o *gpad*

Enquadramento

Padrões de Desenho

Framework Aplicacional

Framework gPad

...

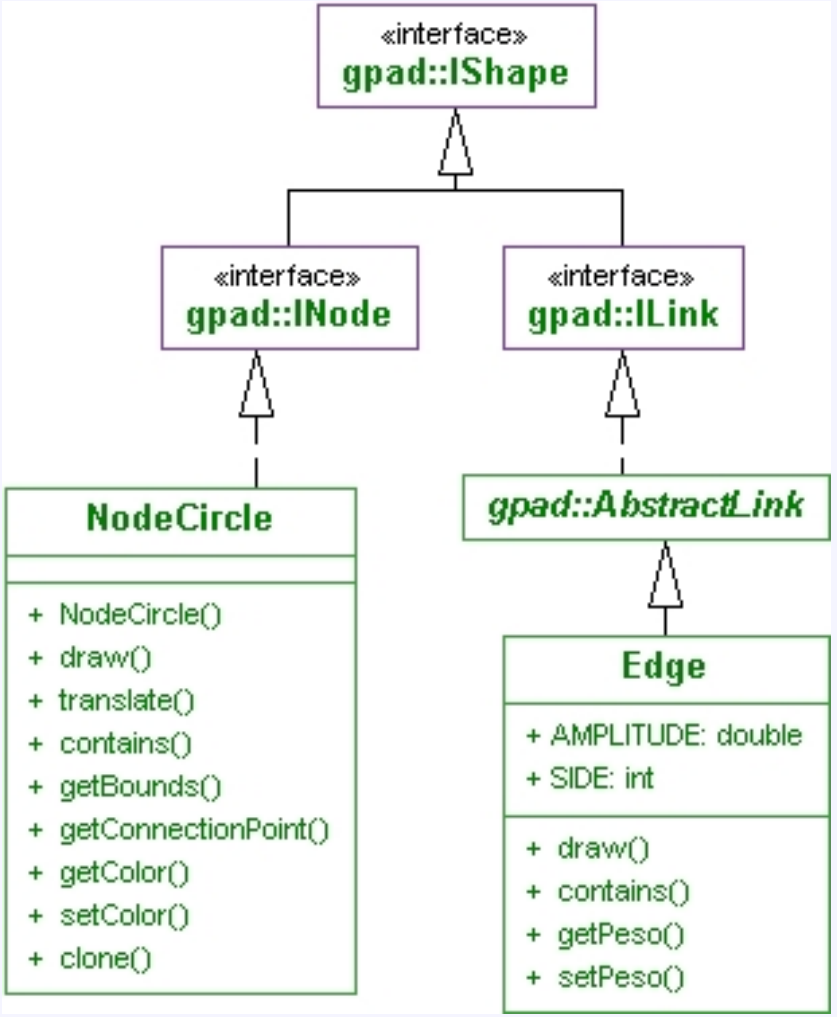
Conclusões

Referências

Editor Simples de Grafos

- Definir uma classe que representa os nós e implementa a interface `INode`;
- Definir uma classe que representa os arcos e implementa a interface `ILink` ou que estende a classe `AbstractLink`.
- Instanciar `PadFrame` com um *array* com dos protótipos correspondente a uma instância de cada tipo de nó e ligação.

```
public class SimpleGraphEditor{
  public static void main(String[] args){
    IShape [] a  = {
      new NodeCircle(Color.BLUE),
      new NodeCircle(Color.WHITE),
      new Edge()
    };
    JFrame frame = new PadFrame(a);
    frame.setTitle("Simple Graph Editor");
    frame.setVisible(true);
  }
}
```



Conclusões

Enquadramento

Padrões de Desenho

Framework Aplicacional

Framework gPad

Aplicações sobre o *gpad*

...

Referências

Conclusões

- **Nunca pensar: "Como é que vou aplicar um padrão para este problema!"**
  ➔ usar um padrão apenas se for aplicável ao problema.
  ➔ se existir uma solução mais simples, então esta deve ser avaliada antes da aplicação de um padrão.
  ➔ no entanto a variância do sistema pode-nos levar a optar por soluções mais complexas, mas que oferecem flexibilidade e extensibilidade.

- **Os padrões de desenho não são fórmulas mágicas**.
  ➔ quando aplicados devem ser analisadas as consequências sobre o resto do desenho.

- A fase de desenho não é a única etapa em que podem ser aplicados padrões.
  ➔ ***Refactoring***: processo de alteração do código de modo a melhorar a estrutura e organização da aplicação, sem modificar o comportamento.

- Não esquecer a importância de saber usar os **princípios fundamentais de OO** para desenhar boas soluções.