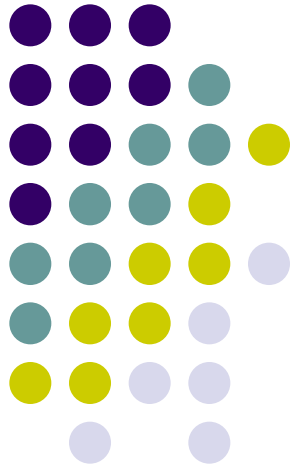
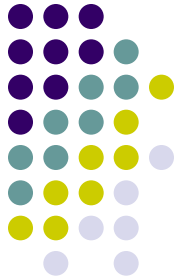
COM220
Aula 13: Introdução ao modelo MVC

Prof. Laércio Baldochi

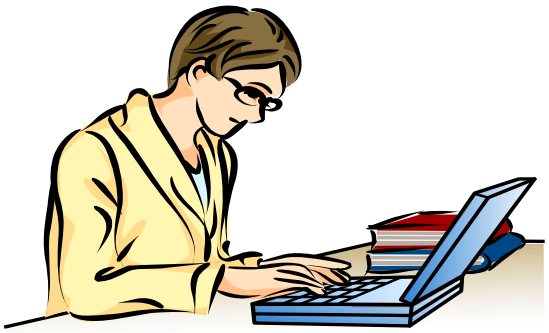


O que é?



- O padrão MVC é um paradigma baseado em eventos que divide um sistema em 3 partes:
  - Modelo (Model)
    - Modelo de dados. Encapsula o estado do sistema
  - Visão (View)
    - Interface do usuário. Provê uma representação visual do modelo
  - Controle (Controller)
    - Controle lógico do sistema. Mapeia ações (realizadas sobre uma dada visão) em mudanças no modelo

Arquitetura MVC



View

Inclui
Informações

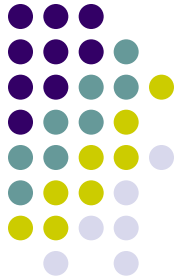
Controller

Altera o
modelo

Model

Disparo de
evento ao
ocorrer alteração
no modelo

Arquitetura MVC

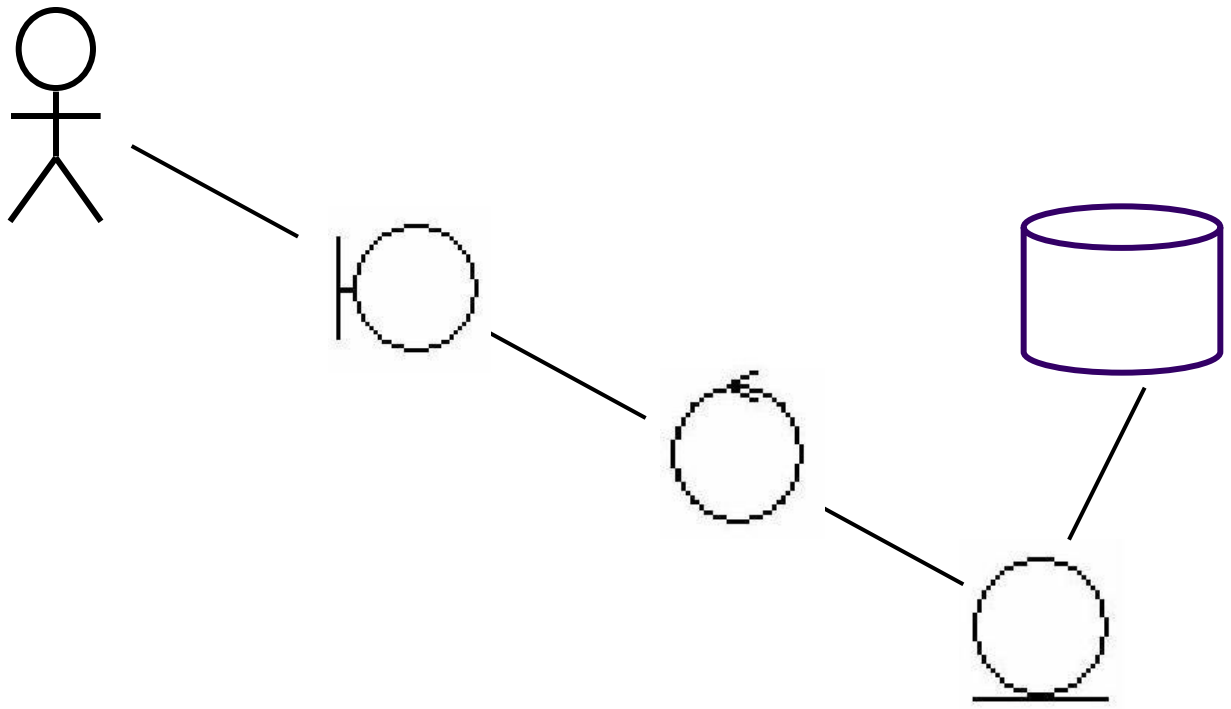


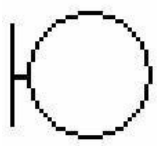
- Lida com a separação de preocupações (separation of concerns)
- Separa
  - dados e/ou lógica de negócio
    - da
  - interface do usuário
    - do
  - controle da aplicação

- Importante
  - O modelo do negócio não deve saber nada sobre as telas que exibem seu estado

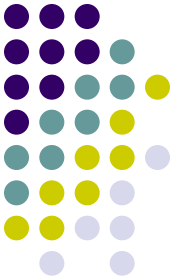
Arquitetura MVC

- A separação de preocupações é alcançada por meio de uma arquitetura a qual propõe a utilização de 3 tipos de classes
  - Limite
  - Controle
  - Entidade

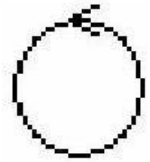




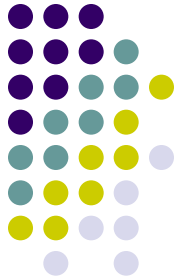
Classe limite



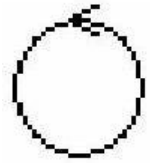
- Representa as relações entre os atores (mundo externo) e o sistema
- Seus objetos traduzem os eventos gerados por um ator em eventos relevantes ao sistema
- Normalmente tem as seguintes responsabilidades
  - Notificar os objetos de controle sobre eventos gerados externamente ao sistema
  - Notificar os atores sobre o resultado de interações entre objetos internos



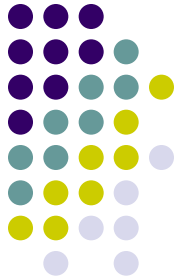
Classe controle (controladora)



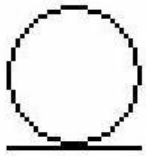
- É responsável pela separação entre as classes de interface (limite) e as classes da lógica de negócio
- Responsável por controlar a lógica de execução correspondente a um caso de uso
- Decide o que o sistema deve fazer quando um evento externo ocorre
  - Traduz eventos externos em operações que devem ser realizadas pelos demais objetos



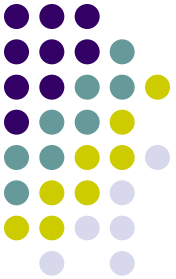
Classe controle (controladora)



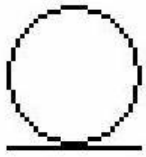
- Suas instâncias têm, normalmente, vida curta
  - Existem somente durante a realização de um caso de uso
- Responsabilidades típicas
  - Realizar monitorações, a fim de responder a eventos externos ao sistema (gerados por objetos limite)
  - Coordenar a realização de um caso de uso, enviando mensagens a objetos limite e objetos entidade
  - Assegurar que as regras de negócio estão sendo cumpridas



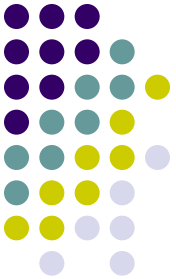
Classe entidade



- São as classes do domínio do problema
  - Em um sistema acadêmico: curso, aluno, professor, ...
- Manipulam informação especializada e encapsulam o conhecimento do negócio
- Na maioria das vezes, manipulam informações persistentes. Por isso, podem ser usadas para gerar diretamente o esquema da base de dados
- Atores nunca têm acesso direto a esse tipo de classe
- Participam de vários casos de uso e têm ciclo de vida longo



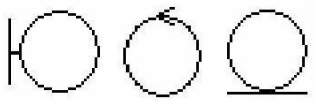
Classe entidade

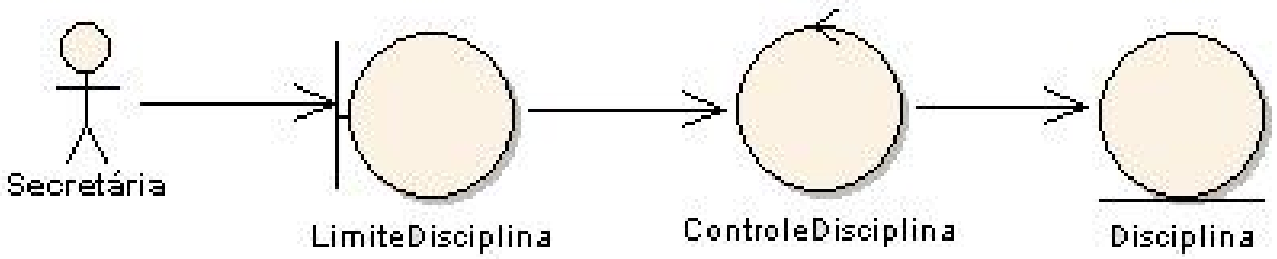


- Responsabilidades típicas
  - Informar valores de seus atributos a objetos de controle
  - Realizar cálculos simples, normalmente com a colaboração de objetos de entidade associados através de agregações
  - No caso de possuir subclasses, criar e destruir objetos

Análise de robustez

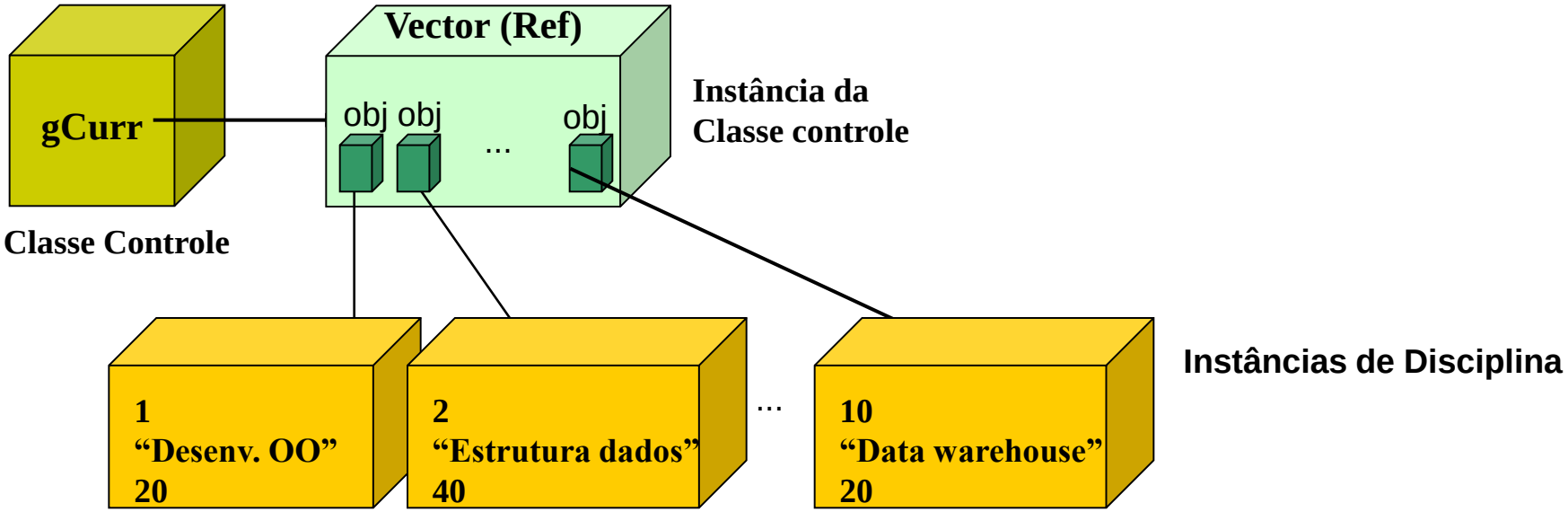
- Proposta por Ivar Jacobson, a partir do padrão MVC
- Fornece uma visão de alto nível de um sistema, na qual é possível identificar as classes limite, de controle e de entidade
- Utiliza representação icônica
- Ex:





Exercício 1

- Com base nos seus conhecimentos sobre a classe Vector e sobre a aplicação dos conceitos de *encapsulamento*, desenvolva um programa em Java que guarde *código*, *nome* e *carga horária* de disciplinas de um determinado curso. Para tal construa uma classe Entidade de nome *Disciplina*, uma classe controle de nome *ControleDisciplina* e uma classe limite *LimiteDisciplina* (que será a classe pública que dará nome ao programa e que conterá a função main()).



Exercício 1
Vejamos os diagramas de robustez e de classe

Diagrama de Robustez

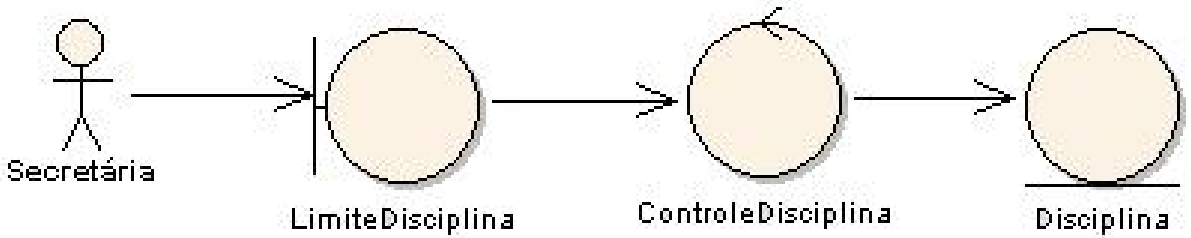
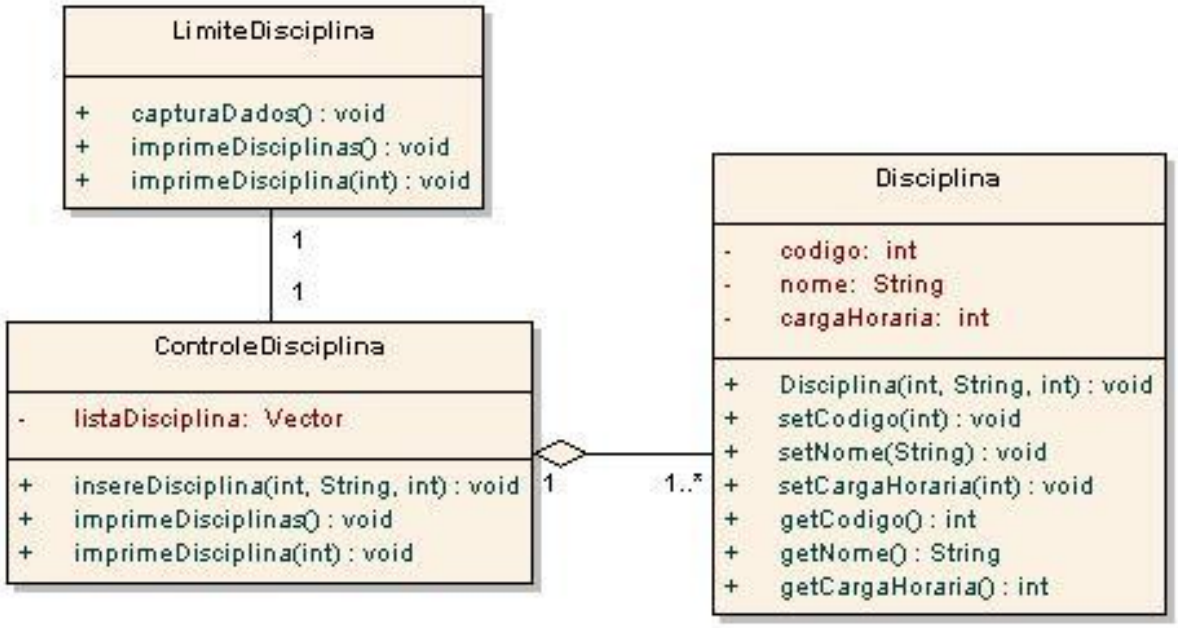
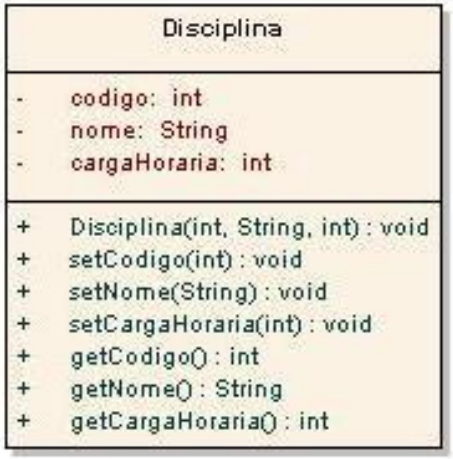


Diagrama de Classes



Implementação

Classe Disciplina



Encapsulamento: Todos os atributos da classe **Disciplina** são definidos como privados "**private**" e somente podem ser acessados através de métodos públicos "**public**".

```
public class Disciplina {
  private int codigo;
  private String nome;
  private int cargaHoraria;

  public Disciplina(int pCodigo, String pNome, int pCargaHoraria) {
    codigo = pCodigo;
    nome = pNome;
    cargaHoraria = pCargaHoraria;
  }
  //Métodos de atribuição
  public void setCodigo(int pCodigo) {
    codigo = pCodigo;
  }
  public void setNome(String pNome) {
    nome = pNome;
  }
  public void setCargaHoraria(int pCargaHoraria) {
    cargaHoraria = pCargaHoraria;
  }
  //Métodos de recuperação
  public int getCodigo() {
    return codigo;
  }
  public String getNome() {
    return nome;
  }
  public int getCargaHoraria() {
    return cargaHoraria;
  }
}
```

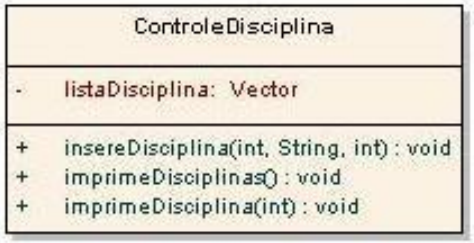
Construtor da classe Disciplina com três argumentos

Métodos de atribuição de informação set...()

Métodos de recuperação de informação get...()

Implementação

Classe ControleDisciplina



Encapsulamento: Todos os atributos da classe **ControleDisciplina** são definidos como privados "**private**" e somente podem ser acessados através de métodos públicos "**puclic**".

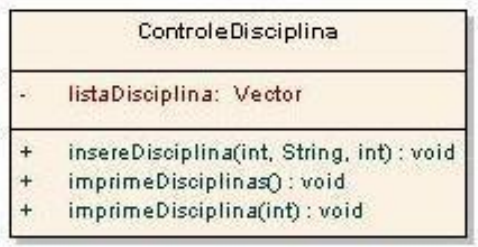
```
import java.util.*;

public class ControleDisciplina {

  private Vector listaDisciplina = new Vector();

  //Método de inserção
  public void insereDisciplina(int pCodigo,
            String pNome, int pCargaHoraria) {
    Disciplina disc = new Disciplina(pCodigo, pNome, pCargaHoraria);
    listaDisciplina.add(disc);
  }

  //Método de impressão da lista de disciplinas
  public String imprimeDisciplinas() {
    String result = "";
   for (int intIdx = 0; intIdx < listaDisciplina.size(); intIdx++) {
    Disciplina disc = (Disciplina)listaDisciplina.elementAt(intIdx);
    result += imprimeDisciplina(disc.getCodigo());
   }
    return result;
 }

....
```

Cria o objeto disc (instância da classe **Disciplina**) utilizando o construtor que recebe parâmetros

Converte o elemento corrente do vetor para um objeto da classe Disciplina (*type cast*). Através desse type cast é possível acessar os métodos da classe **Disciplina.**

Implementação

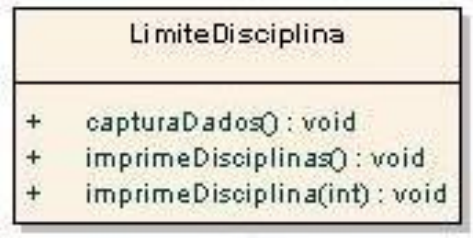
Classe ControleDisciplina




```
//Método utilizado para imprimir uma determinada disciplina
public String imprimeDisciplina(int pCodigo) {
  for (int intIdx = 0; intIdx < listaDisciplina.size(); intIdx++) {
    if (((Disciplina)listaDisciplina.elementAt(intIdx)).getCodigo() ==
          pCodigo) {
      return
        "Código: " +
          ((Disciplina)listaDisciplina.elementAt(intIdx)).getCodigo()+
        "  Nome: " +
          ((Disciplina)listaDisciplina.elementAt(intIdx)).getNome()+
        "  Carga horaria: "+
    ((Disciplina)listaDisciplina.elementAt(intIdx)).getCargaHoraria();
    }
  }
  return "";
 }
}
```

Implementação

Classe LimiteDisciplina



Cria o objeto ctrDisc (instância da classe **ControleDisciplina**)

Chama os métodos da classe **ControleDisciplina** através do objeto ctrDisc

```
import javax.swing.*;

public class LimiteDisciplina  {

  private ControleDisciplina ctrDisc = new ControleDisciplina();
  //Método utilizado para inserir os dados (simulando uma tela de
cadastro)
  public void capturaDados() {
    ctrDisc.insereDisciplina(1, "Desenvolvimento OO", 108);
    ctrDisc.insereDisciplina(2, "Estrutura de dados", 60);
    ctrDisc.insereDisciplina(3, "Data Warehouse", 60);
  }
  //Método utilizado para imprimir a lista de disciplinas (simulando
uma tela de consulta)
  public void imprimeDisciplinas() {
    System.out.println(ctrDisc.imprimeDisciplinas());
  }
  //Método utilizado para imprimir uma disciplina específica
(simulando uma tela de consulta)
  public void imprimeDisciplina(int pCodigo) {
    System.out.println(ctrDisc.imprimeDisciplina(pCodigo));
  }
  //Método principal da classe
  public static void main (String par[]) {
     LimiteDisciplina limDisc = new LimiteDisciplina();
     limDisc.capturaDados();
     limDisc.imprimeDisciplinas();
     System.out.println("--------------------");
     limDisc.imprimeDisciplina(2);
  }
}
```

Implementação

Classe LimiteDisciplina (opcional)

```
import javax.swing.*;
public class LimiteDisciplinaI  {

  public static void main (String par[]) {

    int escolha = 0;
    String escolhaInformada = "";
    //Variáveis utilizadas para recuperar as informações da
    // interface do usuário
    int codigo = 0;
    String nome = "";
    int cargaHoraria = 0;
    String retorno = "";

    //Instancia o Controlador
    ControleDisciplina ctrDisc = new ControleDisciplina();

    do {
        do {
          escolhaInformada =
            JOptionPane.showInputDialog(
                 "Escolha uma opção do menu:\n"+
                 "[1] Adiciona disciplina\n"+
                 "[2] Lista disciplinas\n"+
                 "[3] Finaliza");
          escolha = Integer.parseInt(escolhaInformada);
        } while ((escolha < 1) || (escolha > 3));

      if (escolha == 3) System.exit(0);
....
```

Cria o objeto ctrDisc (instância da classe **ControleDisciplina**)

Implementação

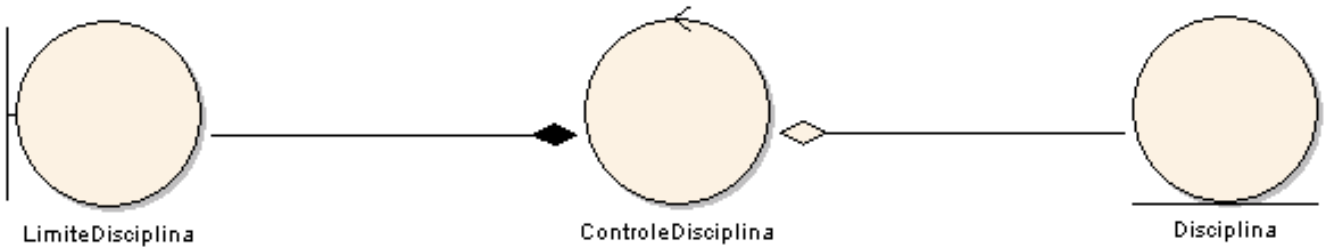
Classe LimiteDisciplina (opcional)

```
....

switch (escolha) {
 case 1:
  //Requisita o Código
  retorno = JOptionPane.showInputDialog ("Informe o código");
  codigo =  Integer.parseInt(retorno);
  //Requisita o Nome
  nome = JOptionPane.showInputDialog ("Informe o nome");
  //Requisita a Carga Horária
   retorno =
     JOptionPane.showInputDialog ("Informe o carga horária");
   cargaHoraria = Integer.parseInt(retorno);
   //Adiciona o objeto para a lista de disciplinas
   ctrDisc.insereDisciplina(codigo, nome, cargaHoraria);
   break;
 case 2:
   JOptionPane.showMessageDialog(null,
     ctrDisc.imprimeDisciplinas(), "Relação de Disciplinas",
     JOptionPane.INFORMATION_MESSAGE);
  }
 } while (true);
}
}
```

Chama os métodos da classe **ControleDisciplina** através do objeto ctrDisc

Exercício 2

- Implementaremos uma nova versão do exercício 1 utilizando uma abordagem mais robusta, na qual o controle cria o limite e o limite apenas interage com o usuário. O diagrama de robustez para essa abordagem é o seguinte:



Implementação

```
import java.util.*;
public class ControleDisciplina {
  private Vector listaDisciplina = new Vector();
  private LimiteDisciplina limDis;

 // Construtor de ControleDisciplina
ControleDisciplina( ) {
    limDis = new LimiteDisciplina(this);
  }
 ....
}
```

```
import javax.swing.*;
public class LimiteDisciplina() {
   ControleDisciplina ctrDisc;
   ...
public LimiteDisciplina (ControleDisciplina ctrDiscPar)  {
    ctrDisc = ctrDiscPar;
  ...
}
```

```
import java.util.*;
public class Exercicio2{
 void public main (String par[]) {
    ControleDisciplina ctrDisc = new ControleDisciplina();
     ....
 }
}
```

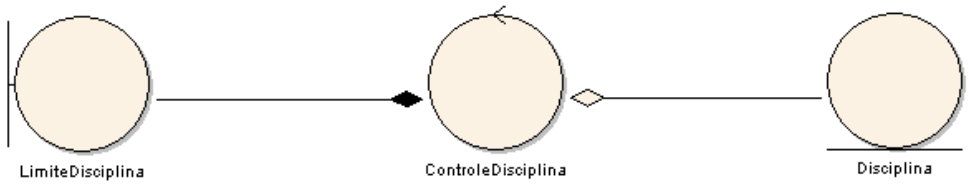
A classe ***ControleDisciplina*** possui um objeto da classe ***LimiteDisciplina***

O objeto ***ControleDisciplina*** é encaminhado ao construtor de ***LimiteDisciplina( )***

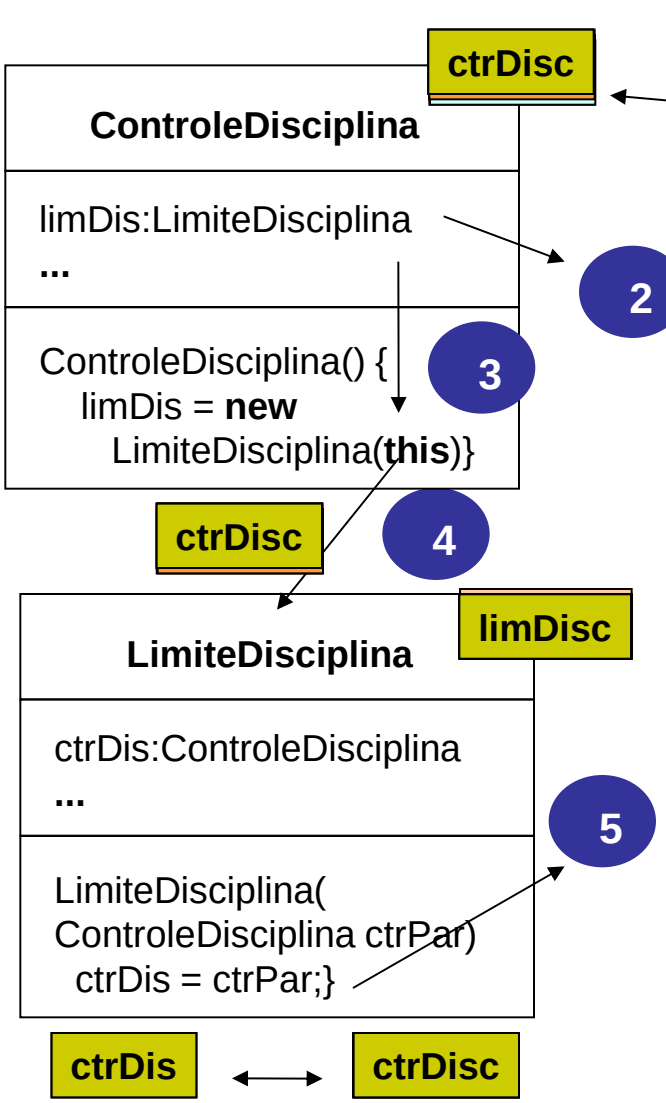
A função main() cria um Controle, que é capaz de desencadear a criação dos limites

A classe LimiteDisciplina tem construtor capaz de receber a instância do controlador

1 2 3 4 5



Implementação




```
import java.util.*;
public class Exercicio2{
 void public main (String par[]) {
    ControleDisciplina ctrDisc = new ControleDisciplina();
     ....
 }
}
```

1 – função main() cria o objeto de *ControleDisciplina ctrDisc*

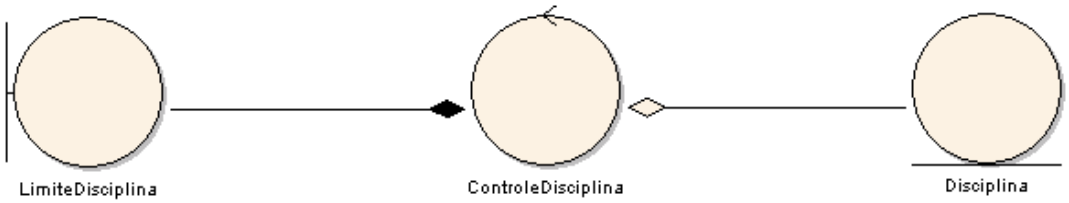
2 – Na criação de *ctrDisc* o objeto de *LimiteDisciplina limDis* é inicializado (com null). Esse objeto é atributo da classe *ControleDisciplina*

3 – Em seguida é chamado o construtor de *ControleDisciplina*, onde o objeto *limDis* será criado. Para essa criação, é encaminhado ao construtor da classe *LimiteDisciplina* justamente o objeto de *ControleDisciplina* que está em ação (ponteiro **this**)

4 – A criação do objeto de *LimiteDisciplina* chamará a inicialização do objeto

5 – Em seguida, chamará o construtor da classe *LimiteDisciplina* que fará a alocação do objeto de *ControleDisciplina* que veio do main() para o atributo de *LimiteDisciplina*.

Vantagens da nova abordagem



A relação Limite-Controle colocada no exercício 2 é a mais adequada para projetos Orientados a Objetos, porque permite que a mudança de regras de interface estejam completamente separadas das regras de negócio do sistema, que ficam implementadas ao nível das classes de Controle e Entidade.

Um exemplo é a troca da biblioteca de interfaces do sistema (ex: trocar AWT por Swing, ou vice-versa). Na relação Limite-Controle realizada no exercício anterior, você poderia realizar essa troca sem que tenha que rever as regras que implementou nas classes de controle e de limite.

Outro exemplo é o desenvolvimento de uma interface Web, que poderia ser desenvolvida para as novas classes Limite, mantendo sua relação com as regras de negócio nas classes de Controle e de Entidade.

Implementação – exercício 2

```
package AulaPOO;

public class Disciplina {
        private  int codigo;
        private String nome;
        private int cargaHoraria;

         /* Construtor da entidade Disciplina.
         * @param pCodigo código da disciplina.
         * @param pNome descrição do nome da disciplina
         * @param pCargaHoraria carga horária da disciplina
        */
        public Disciplina(int pCodigo, String pNome,
                    int pCargaHoraria) {
           codigo = pCodigo;
           nome = pNome;
           cargaHoraria = pCargaHoraria;
        }
        /**
         * Atribui o código da disciplina.
         * @param pCodigo código da disciplina.
         */
        public void setCodigo(int pCodigo) {
           codigo = pCodigo;
        }
        /**
         * Atribui a descrição do nome da disciplina.
         * @param pNome nome da disciplina.
         */
        public void setNome(String pNome) {
           nome = pNome;
        }
        /**
         * Atribui a carga horária da disciplina.
         * @param pCargaHoraria carga horária da disciplina.
         */
        public void setCargaHoraria(int pCargaHoraria) {
           cargaHoraria = pCargaHoraria;
        }

        /**
         * Informa o código da disciplina.
         * @return int código da disciplina.
         */
        public int getCodigo() {
           return codigo;
        }

        /**
         * Informa o nome da disciplina.
         * @return String nome da disciplina.
         */
        public String getNome() {
           return nome;
        }

        /**
         * Informa a carga horária da disciplina.
         * @return int carga horária da disciplina.
         */
        public  int getCargaHoraria() {
           return cargaHoraria;
        }
}
```

Implementação – exercício 2

```
package AulaPOO;

import java.util.Vector;

public class ControleDisciplina {
        private Vector listaDisciplina = new Vector();
        private LimiteDisciplina objCLimiteDisc;

        /**
         * Construtor da classe.
         */
        public ControleDisciplina() {
             objCLimiteDisc = new LimiteDisciplina(this);
        }

        /**
         * Método responsável por fazer a inserção de uma
         * disciplina no vetor.
         * @param pCodigo código da disciplina.
         * @param pNome descrição do nome da disciplina.
         * @param pCargaHoraria carga horária da disciplina.
         */
        public void insereDisciplina(int pCodigo,
                    String pNome, int pCargaHoraria) {
           Disciplina disc = new Disciplina(pCodigo, pNome,
                    pCargaHoraria);
           listaDisciplina.add(disc);
        }
```

```
/**
 * Método responsável por preparar um texto
 * contendo todas as informações
 * de disciplinas cadadastradas para serem
 * apresentadas ao usuário.
 * @return String descrições de todas as disciplinas
 *    cadastradas.
 */
public String imprimeDisciplinas() {
     String result = "";
     for (int intIdx = 0; intIdx < listaDisciplina.size();
              intIdx++) {
           Disciplina objLDisc =
                  (Disciplina)listaDisciplina.elementAt(intIdx);
           result += "Código: " +
              objLDisc.getCodigo() +
   "  Nome: " + objLDisc.getNome()+
              "  Documento: " +
              objLDisc.getCargaHoraria() + "\n";
     }
     return result;
}
```

Implementação – exercício 2

```
/**
 * Método responsável por preparar a apresentação
 * das informações de uma determinada
 * disciplina.
 * @param pCodigo código da disciplina para
 *     apresentação de suas respectivas
 *     informações.
 * @return String descrição de todas as
 *     informações da disciplina
 */
public String imprimeDisciplina(int pCodigo) {
   for (int intIdx = 0; intIdx < listaDisciplina.size();
            intIdx++) {
       Disciplina objLDisc =

(Disciplina)listaDisciplina.elementAt(intIdx);
       if (objLDisc.getCodigo() == pCodigo) {
           return "Código: " + objLDisc.getCodigo()+
                                "  Nome: " +
objLDisc.getNome()+
                                "  Documento: " +
                                  objLDisc.getCargaHoraria();
      }
   }
   return "";
}
```

```
/**
 * Método principal responsável apenas por criar um
 * objeto da classe ControleDisciplina.
 * @param args[] argumentos passados por
 *     parâmetros. Nesse sistema nenhum
 *     argumento é esperado.
 */
public static void main(String args[]) {
          new ControleDisciplina();
}
}
```

Implementação – exercício 2

```
package AulaPoo;

import javax.swing.JOptionPane;

public class LimiteDisciplina {
      private ControleDisciplina ctrDisc;

      /**
       * Construtor da classe.
       * @param objPControleDisc é uma referência do objeto
       *     ControleDisciplina.
       */
      public LimiteDisciplina( ControleDisciplina
                                                objPControleDisc) {
        ctrDisc = objPControleDisc;
        CapturaDados();
      }
```

```
/**
 * É responsável por gerenciar o menu principal da
 * aplicação.
 */
private void CapturaDados() {
   int escolha;
   do {
      do {
        String escolhaInformada =
           JOptionPane.showInputDialog(
            "Escolha uma opção do menu:\n" +
            "[1] Adiciona disciplina\n" +
            "[2] Lista uma disciplina\n" +
            "[3] Lista disciplinas\n" +
            "[4] Finaliza");
          Escolha =
                    Integer.parseInt(escolhaInformada);
        } while ((escolha < 1) || (escolha > 4));
        if (escolha != 4) {
           execEscolha(escolha);
        } else {
           System.exit(0);
        }
   } while (true);
}
```

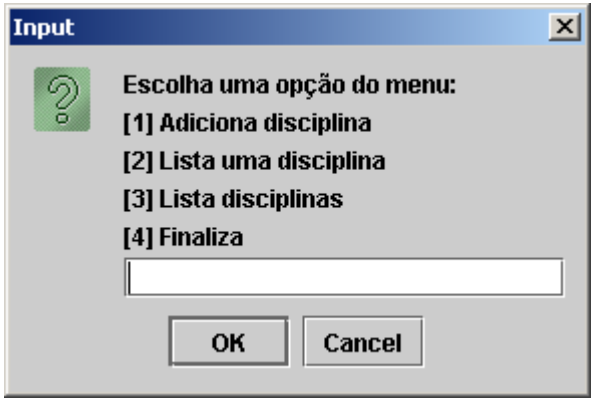
Implementação – exercício 2

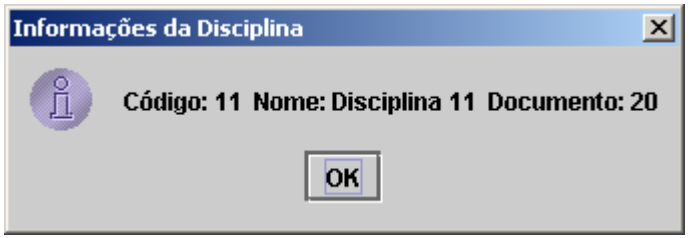
```
/**
 * Faz o tratamento da escolha da opção do
 * usuário.
 * @param intPEscolha opção escolhida pelo
 *     usuário.
 */
private void execEscolha(int intPEscolha) {
          switch (intPEscolha) {
             case 1: {
                cadastraDisciplina(); break;
             }
              case 2: {
                 int intLCodigo = Integer.parseInt(
                      JOptionPane.showInputDialog(
                      "Digite o código da disciplina"));
                 imprimeDisciplina(intLCodigo);
                 break;
              }
               case 3: {
                  imprimeDisciplinas();
               }
          }
}
```

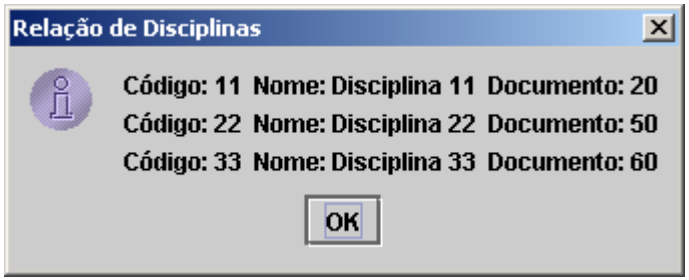
```
/**
 * Método responsável por fazer a chamada para
 * cadastrar uma disciplina.
 */
private void cadastraDisciplina() {
   String retorno = JOptionPane.showInputDialog(
          "Informe o código");
   int codigo =  Integer.parseInt(retorno);
   String nome = JOptionPane.showInputDialog(
          "Informe o nome");
   retorno = JOptionPane.showInputDialog (
          "Informe o carga horária");
   int cargaHoraria = Integer.parseInt(retorno);
   ctrDisc.insereDisciplina(codigo, nome,
          cargaHoraria);
}
```

Implementação – exercício 2

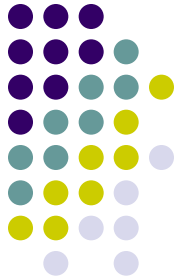
```
    /**
     * Método reponsável por apresentar as
     * informações das disciplinas.
     */
    public void imprimeDisciplinas() {
       JOptionPane.showMessageDialog(null,
              ctrDisc.imprimeDisciplinas(),
              "Relação de Disciplinas",
              JOptionPane.INFORMATION_MESSAGE);
    }

    /**
     * Método responsável por apresentar as
     * informações de uma única disciplina.
     * @param pCodigo código de uma disciplina
     */
    public  void imprimeDisciplina(int pCodigo) {
       JOptionPane.showMessageDialog(null,
              ctrDisc.imprimeDisciplina(pCodigo),
              "Informações da Disciplina",
              JOptionPane.INFORMATION_MESSAGE);
    }
}
```





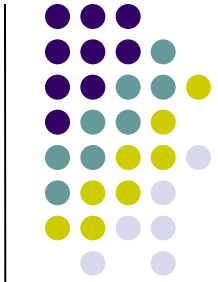


Exercício 3
Entrega: hoje



- Baseado no exercício anterior, implemente um cadastro semelhante para estudantes. Cada estudante deve possuir código, nome e endereço. Os estudantes devem ser armazenados num objeto do tipo *Vector*. O sistema deve apresentar um menu disponibilizando opções para cadastro, remoção e visualização de informações dos estudantes.
- É necessário elaborar os diagramas de robustez e de classes antes da implementação.

Créditos



- Os exemplos de código apresentados nessa aula foram elaborados por Roberto Pacheco da UFSC