LPI - Tutorial 3 : Classes referenciando outras Classes

1 - Idealizando uma Aplicação para Exercitar o Mapeamento de Entidades do Mundo Real

No Tutorial 1, comentamos que a capacidade de mapear entidades do mundo real em novos tipos, para serem utilizados na aplicação que você pretende desenvolver, é um dos grandes motivos que contribui para a enorme popularização do Paradigma Orientado a Objetos (POO). Neste tutorial, você vai ter a oportunidade de avaliar melhor essa vantagem de POO, através de um projeto ilustrativo que mapeia várias entidades do mundo real em classes da sua aplicação.

Neste tutorial, você será orientado a projetar, no projeto LPI-E3, uma aplicação com várias entidades na linguagem Java, e constatar como é útil mapear entidades do mundo real em novos tipos de uma aplicação em desenvolvimento. Vamos ilustrar os passos desse processo com o projeto de uma aplicação simples. Mas o aprendizado é assim, precisamos começar com projetos simples. Na próxima disciplina, Linguagem de Programação II, você aprenderá a construir aplicações desktop (que podem ser instaladas na sua máquina) utilizando interfaces gráficas e um banco de dados relacional. E finalmente, na disciplina seguinte, Linguagem de Programação III, você aprenderá a construir aplicações que podem ser acessadas a partir de um link na web.

A aplicação escolhida como projeto ilustrativo é uma Escola de Informática. Observe que em uma universidade, você se matricula em disciplinas de um curso, mas em uma escola de informática, não utilizamos o termo disciplina, mas simplesmente curso.

Suponha agora que você já se formou e resolveu montar uma escola de informática. Mas você está querendo oferecer cursos que capacitem os alunos a trabalhar no mercado de trabalho de informática. Sendo assim, você resolveu disponibilizar os seguintes cursos:

- Java Básico - Java Aplicações Locais - Java Aplicações Web
- Banco de Dados Básico - Banco de Dados Avançado

Mas para que os alunos da sua escola possam aproveitar efetivamente os cursos, eles precisam fazer os cursos em uma sequênca adequada. Então, você resolveu adotar os seguintes pré-requisitos:

- Java Básico --- conceitos básicos de POO e de Java
  - sem pré-requisitos
- Java Aplicações Desktop --- aplicações que podem ser instaladas na sua máquina
  - pré-requisitos : Java Básico -- Banco de Dados Básico
- Java Aplicações Web --- aplicações que são acessadas a partir de links da web
  - pré-requisitos : Java Aplicações Desktop
- Banco de Dados Básico --- projeto de bases de dados, e utilização de SQL para criação, pesquisas e alterações em bases de dados relacionais
  - sem pré-requisitos
- Banco de Dados Avançado --- projeto e utilização de grandes bases de dados
  - pré-requisitos : Banco de Dados Básico

Mas é claro que como você se formou em Sistemas de Informação, você mesmo vai projetar e implementar o software para matricular alunos na sua escola de informática.

Agora que você já sabe o que pretende fazer, vamos ao primeiro passo:

- descrever, em alto nível, os requisitos que a sua aplicação deve suportar.

2 - Primeiro Passo : Especificando os Requisitos da Aplicação

Os requisitos essenciais da sua aplicação podem ser descritos de uma forma bem amigável:

- a aplicação deve suportar a matrícula de alunos nos cursos da escola de informática, com os seguintes dados: quem é o aluno matriculado, em qual curso, e em que data;
- mas as matrículas devem referenciar alunos e cursos previamente cadastrados na aplicação;
- para incentivar as matrículas, a escola de informática deverá oferecer algumas bolsas, para alunos que prestarem uma prova de avaliação, nas seguintes categorias:
  - bolsa completa : aluno não paga o curso;
  - bolsa parcial : aluno paga metade do valor do curso;
  - nenhuma: aluno paga o valor integral do curso.
- a matrícula em um curso, depende de que o aluno tenha sido aprovado previamente nos cursos que são pré-requisitos para esse curso;
  - para ser aprovado em um curso o aluno precisa atingir a nota final 7,0;
- no cadastro do aluno deverão constar, além de seus dados principais, os seus dados de contato, endereço e a sua data de nascimento, para parabenizá-lo quando ele fizer aniversário.

Agora que você já definiu os requisitos da sua aplicação, vamos ao segundo passo:

- olhar para o mundo real, e mapear as entidades de interesse como novos tipos da sua aplicação, ou seja, suas classes entidade.

3 - Segundo Passo : Mapeando as Entidades em Classes com Atributos e Referências

O mundo real oferece infinitas possibilidades para o desenvolvimento de aplicações. Depois que você definiu os requisitos da suas aplicações, você deverá identificar as entidades do mundo real que precisará mapear na sua aplicação, para atender os requisitos que você estabeleceu.

Bem, você já sabe que deseja construir uma aplicação para suportar matrículas de alunos em cursos. Então você precisará mapear as seguintes entidades do mundo real, que são essenciais para a sua aplicação: Matrícula, Aluno e Curso.

Mas você também deseja armazenar em seus cadastros: a data da matrícula e do aniversário do aluno, bem como, o endereço do aluno.

Um endereço tem vários atributos (ex: rua, numero, complemento, bairro, cidade, cep) e se aplica a muitas situações, tais como: endereços de clientes, empresas, amigos, etc. Sobrecarregar a classe Aluno, com vários atributos específicos de endereço, decididamente não é uma boa idéia. Então vamos criar a classe genérica Endereço, que poderá ser reutilizada em inúmeras aplicações.

E quanto ao cadastro de datas? Armazenar uma data como um string do tipo "dia/mês/ano", também não é uma boa idéia. Se você criar uma classe genérica Data, com os atributos dia, mês e ano, você poderá simplificar as implementações de métodos: para comparar duas datas (para saber se as duas datas são iguais ou uma delas é anterior ou posterior à outra) e para calcular a idade de uma pessoa (que não depende somente do ano de nascimento, mas também do mês e do dia). Vamos criar a classe genérica Data com esses métodos que são muito úteis.

Portanto, nossa aplicação deverá suportar as seguintes entidades, mapeadas em classes: Matrícula, Aluno, Curso, Endereço e Data.

Para definirmos os atributos das classes de entidade, vamos pensar de forma top-down, ou seja, de cima para baixo. O principal objetivo, da sua aplicação é suportar matrículas. Para organizar adequadamente a informatização de uma escola de informática, o cadastro uma matrícula implica no cadastro prévio dos cursos oferecidos pela escola e dos alunos que frequentarão a escola. Os dados de uma matrícula deverão caracterizar: aluno matriculado, o curso no qual o aluno se matriculou, a data da matrícula e a o tipo de bolsa associada à matrícula. Portanto, a classe Matrícula deverá definir: (a) o atributo bolsa; e (b) as referências aluno, curso e data_início. Atributos caracterizam propriedades de objetos de uma classe. Referências caracterizam objetos de outras classes referenciados pelos objetos de uma classe.

O dados dos cursos cadastrados deverão caracterizar: o título do curso, sua carga horária semanal e sua duração_semanas. Adicionalmente, o curso deverá caracteriar os títulos dos cursos que são pre-requisitos para se matricular nele. Portanto a classe Curso deverá definir os atributos: título, carga_horária_semanal, duração_semanas e pré-requisitos (lista de títulos dos cursos pré-requisitos).

Os dados dos alunos cadastrados deverão caracterizar: o nome do aluno, seu RG, CPF, data de nascimento, sexo e endereço. Portanto, a classe Aluno deverá definir: (a) os atributo nome, rg, cpf e sexo; e (b) as referências data_nascimento e endereço.

A classe Endereço deverá definir pelo menos: logradouro (rua, avenida, praça, etc), número, completo (apto ou fundos, por exemplo), bairro, cidade, cep. Dependendo da aplicação, poderá ser interessante definir o estado (ou província ou região) e o país. A classe Data é bem padronizada e deverá definir os atributos: dia, mês e ano.

4 - Terceiro Passo : Implementando as Classes de Entidade e Classes Auxiliares

Após definirmos os atributos e referências das classes, vamos implementar cada uma das classes mapeadas a partir das entidades do mundo real relevante para a nossa aplicação. Para que o PyCharm não fique acusando classes indefinidas, é aconselhável iniciar pela definição das classes que não tem referência para nenhum outra classe (Data, Endereço e Curso), para finalizar com as classe que referenciam objetos de classes previamente definidas (Aluno e Matrícula).

Para cada módulo utilizado, vamos apresentar a implementação completa das classes e esclarecer, sempre que necessário, a utilidade e a implementação dos métodos e das funções definidas no respectivo módulo.

É praticamente uma convenção entre as linguagens de POO, a definição de classes auxiliares que forem úteis para aplicações em geral em um diretório denominado util. Portanto, definiremos a classe Data no módulo data do diretório util (embora seja acentuado em português, utilizaremos o nome em inglês, consagrado nas linguagens de POO). A seguir, a implementação do módulo data, com a classe auxiliar Data e uma função associada a ela.

```
from datetime import date

def str_to_data(data_str):
    dia, mês, ano = data_str.split('/')
    return Data(int(dia), int(mês), int(ano))

class Data:

    def __init__(self, dia, mês, ano):
        self.dia = dia
        self.mês = mês
        self.ano = ano

    def __str__(self):
        if self.dia < 10: data_str = '0' + str(self.dia)
        else: data_str = str(self.dia)
        if self.mês < 10: data_str += "/0" + str(self.mês) + "/"
        else: data_str += "/" + str(self.mês) + "/"
        data_str += str(self.ano)
        return data_str

   def __eq__(self, data):
        if str(self) == str(data): return True
        return False

   def __ne__(self, data):
        return not self == data

   def __gt__(self, data):
        if self.ano > data.ano: return True
        elif self.ano < data.ano: return False
        if self.mês > data.mês: return True
        elif self.mês < data.mês: return False
        if self.dia > data.dia: return True
        elif self.dia < data.dia: return False
        return False

   def __lt__(self, data):
        if self.ano < data.ano: return True
        elif self.ano > data.ano: return False
        if self.mês < data.mês: return True
        elif self.mês > data.mês: return False
        if self.dia < data.dia: return True
        elif self.dia > data.dia: return False
        return False

   def __ge__(self, data):
        if self < data: return False
        else: return True

   def __le__(self, data):
        if self > data: return False
        else: return True

   def calcular_idade(self):
        dia_atual_str, mês_atual_str, ano_atual_str = date.today().strftime("%d/%m/%Y").split('/')
        dia_atual, mês_atual, ano_atual = int(dia_atual_str), int(mês_atual_str), int(ano_atual_str)
        idade = ano_atual - self.ano
        if mês_atual < self.mês or (mês_atual == self.mês and dia_atual < self.dia): idade -= 1
        return idade
```

Na implementação do método __str__ está sendo checado se o dia ou mês tem apenas um dígito decimal, para gerar um string iniciando com o caracter '0'. Os strings gerados para representar dia, mês e ano estão sendo separados pelo caracter '/' para gerar o string completo da data.

Para que objetos de uma classe possam ser comparados, você pode implementar métodos padronizados de comparação. Na classe Data, este possibilidade é muito útil, para que você possa comparar se dois objetos da classe Data tem a mesma data, ou se a data de uma deles é superior ou inferior à data do outro.

Para podermos verificar se dois objetos são iguais utilizando o comparador de igualdade ==, precisamos implementar o método padronizado __eq__. Desta forma, a comparação entre duas datas, if data1 == data2 é executada como se fosse if data1.__eq__(data2). Data equivalentes devem ter os mesmos dias, meses e anos e, portanto, a primeira ideia que vem a mente seria implementar o método __eq__ da seguinte forma:

```
def __eq__(self, data):
    if self.dia == data.dia and self.mês == data.mês and self.ano == data.ano: return True
    return False
```

Essa implementação parece perfeita, mas não funciona para a comparação: if data == None. No entanto, utilizar o comparador == na implementação do método __eq__ para comparar data com None não pode ser feita dentro do próprio método __eq__ (equal) porque geraria um loop infinito de chamadas recursivas do próprio método __eq__, uma vez que esta método é chamado toda vez que se utiliza o comparador == para comparar um objeto da classe Data. Observe que chamada recursiva é um mecanismo muito poderoso, mas se não for utilizado com critério pode conduzir a um loop infinito e causa erro de execução. A solução encontrada foi comparar o string do objeto da classe Data, porque a comparação de strings não chama o método __eq__ da classe Data e, portanto, também funciona quando a comparação é feita com None.

Para podermos verificar se dois objetos são diferentes utilizando o comparador de desigualdade !=, precisamos implementar o método padronizado __ne__ (not equal). Agora já podemos utilizar o comparador == na implementação de __ne__ por que sua chamada vai gerar a chamada do método __eq__ e, portanto, não haverá chamada recursiva. Desta forma, sua implementação fica muito simplificada.

Para completar precisamos implementar os seguintes métodos padronizados de comparação:

- __gt__ (greater than : maior que) para > ;
- __ge__ (greater or equal: maior ou igual) para >= ;
- __lt__ (lesser than : menor que) para < ;
- __le__ (lesser or equal : menor ou igual) para <= .

Obviamente não faz sentido comparar se uma data é maior ou menor que None e, portanto, podemos utilizar a comparação de dias, meses e anos para implementar esses quatro métodos padronizados de comparação, simplificando as implementações. Para os métodos __gt__ e __lt__, inicialmente comparamos os anos, se forem iguais, comparamos os meses e se forem iguais comparamos os dias. As implementações dos métodos __ge__ e __le__ ficam bem simples, porque se utilizam das implementações de __gt__ e __lt__.

Adicionalmente, é importante implementar o método calcular_idade a partir da data de nascimento de uma pessoa. Inicialmente obtemos os strings correspondentes ao dia, mês e ano da data atual e convertemos esses strings para valores inteiros. Inicialmente subtraímos o ano atual do ano da data de nascimento. Se a pessoa ainda não fez aniversário neste ano, basta subtrairmos 1 deste valor. Caso contrário, este valor já corresponderá à idade da pessoa.

A seguir, veremos a implementação da classe Endereço. Esta classe pode ser utilizada para se associar à objetos de várias outras classes, dado que qualquer pessoa e empresa tem endereço. No entanto, sua implementação tem uma certa dependência da aplicação. Nesta implementação não está sendo armazenada a informação de país, porque supomos que todos os endereços estão associados ao Brasil, mas em outras aplicações o país pode ser relevante. Por esse motivo, vamos definir a classe Endereço no módulo endereço do diretório entidades e não no diretório util.

```
class Endereço:

    def __init__(self, logradouro, número, complemento, bairro, cidade, cep):
        self.logradouro = logradouro
        self.número = número
        self.complemento = complemento
        self.bairro = bairro
        self.cidade = cidade
        self.cep = cep

    def __str__(self):
        endereço_str = self.logradouro + ' - ' + str(self.número)
        if self.complemento != None: endereço_str += ' - ' + self.complemento
        endereço_str += ' - bairro: ' + self.bairro + ' - ' + self.cidade + ' - CEP: ' + self.cep
        return endereço_str
```

A seguir veremos a implementação, no módulo curso do diretório entidades: da classe Curso, do dicionário cursos e das funções get_cursos e inserir_curso. Lembre-se que, conforme vimos no Tutorial 2, que a variável global para armazenar os cursos cadastrados, bem como as funções que a manipulam, estão sendo definidas no módulo no qual foi definida a classe, a partir da qual são criados os objetos.

```
cursos = {}

def get_cursos(): return cursos

def inserir_curso(curso):
    título_curso = curso.título
    if título_curso not in cursos.keys(): cursos[título_curso] = curso
    else: print('Curso já cadastrado')
```

```
class Curso:

    def __init__(self, título, carga_horária_semanal, duração_semanas):
        self.título = título
        self.carga_horária_semanal = carga_horária_semanal
        self.duração_semanas = duração_semanas
        self.pré_requisitos = []

    def __str__(self):
        curso_str = self.título + ' -- carga horária semanal: ' + str(self.carga_horária_semanal)
            + ' - duração em semanas: ' + str(self.duração_semanas)
        for índice, pré_requisito in enumerate(self.pré_requisitos):
            if índice == 0: curso_str += '\n            pré-requisitos: '
            else: curso_str += ' - '
            curso_str += pré_requisito
        return curso_str

    def inserir_pré_requisito(self, pré_requisito):
        if pré_requisito not in self.pré_requisitos: self.pré_requisitos.append(pré_requisito)

    def remover_pré_requisito(self, pré_requisito):
```

```
        if pré_requisito in self.pré_requisitos: del self.pré_requisitos[pré_requisito]
```

Na classe Curso, o atributo pré_requisitos deve ser inserido após o cadastro dos cursos da escola de informática. No método __str__ os títulos dos pré-requisitos serão concatenados no string que representa o objeto da classe Curso. Ao iniciar a concatenação, aparecerá o string 'pré-requisitos' que antecederá somente o primeiro pré-requisito a ser concatenado, ficando os demais separados por ' - '. Foram acrescentados os métodos inserir_pré_requisito para garantir que não será inserido nenhum título duplicado e o método remover_pré_requisito para remover um título de pré-requisito previamente cadastrado.

A classe Aluno é definida no módulo aluno do diretório entidades. Essa classe referencia um objeto de classe Endereço, previamente definida. Observe, que quando seu método __str__ utiliza a função str para converter self.endereço para string, implicitamente o método __str__ da classe Endereço é chamado para retornar o string correspondente ao objeto da classe Endereço referenciado por self.endereço. Adicionalmente vamos definir, no módulo aluno, o dicionário alunos e as funções get_alunos e inserir_aluno.

```
alunos = {}

def get_alunos(): return alunos

def inserir_aluno(aluno):
    cpf_aluno = aluno.cpf
    if cpf_aluno not in alunos.keys(): alunos[cpf_aluno] = aluno
    else: print('Aluno já cadastrado')
```

```
class Aluno:

    def __init__(self, nome, rg, cpf, data_nascimento, sexo, endereço):
        self.nome = nome
        self.rg = rg
        self.cpf = cpf
        self.data_nascimento = data_nascimento
        self.sexo = sexo if sexo in ('M', 'F') else 'indefinido'
        self.endereço = endereço

    def __str__(self):
        return self.nome + ' - RG:' + self.rg + ' - CPF:' + self.cpf + ' - nascimento:'\
            + str(self.data_nascimento) + ' - sexo:' + self.__to_str_sexo__()\
             + '\n          - residente em:: ' + str(self.endereço)

    def __to_str_sexo__(self):
        if self.sexo == 'M': return 'masculino'
        elif self.sexo == 'F': return 'feminino'
        else: return 'indefinido'
```

A última entidade a ser definida é a classe Matrícula, no módulo matrícula do diretório entidades. Adicionalmente, são definidas no módulo matrícula, a lista matrículas e as funções get_matrículas, inserir_matrícula e selecionar_matrículas.

```
from entidades.aluno import get_alunos
from entidades.curso import get_cursos

matrículas = []

def get_matrículas(): return matrículas
```

```
def inserir_matrícula(cpf_aluno, título_curso, bolsa, data_início):
    aluno = alunos[cpf_aluno]
    curso = cursos[título_curso]
    if aluno == None:
        print('Matrícula mal sucedida: aluno não cadastrado')
        return
    if curso == None:
        print('Matrícula mal sucedida: curso não cadastrado')
        return
    matrícula = Matrícula(aluno, curso, bolsa, data_início)
    if matrícula not in matrículas: matrículas.append(matrícula)
    else: print('Matrícula já cadastrada --- ' + str(matrícula))

def selecionar_matrículas(bolsa_matrícula = None, cidade_aluno = None,
    curso_sem_pré_requisitos = None):
    filtros = 'Filtros: '
    if bolsa_matrícula != None: filtros += 'bolsa da matrícula: ' + bolsa_matrícula
    if cidade_aluno != None: filtros += ' - cidade do aluno: ' + cidade_aluno
    if curso_sem_pré_requisitos == True: filtros += ' - curso sem pré-requisitos'
    elif curso_sem_pré_requisitos == False: filtros += ' - curso com pré-requisitos'
    matrículas_selecionadas = []
    for matrícula in matrículas:
        if bolsa_matrícula != None and matrícula.bolsa != bolsa_matrícula: continue
        if cidade_aluno != None and matrícula.aluno.endereço.cidade != cidade_aluno: continue
        matrícula_curso_sem_pré_requisitos = (len(matrícula.curso.pré_requisitos) == 0)
        if curso_sem_pré_requisitos in (True, False)
            and matrícula_curso_sem_pré_requisitos != curso_sem_pré_requisitos: continue
        matrículas_selecionadas.append(matrícula)
    return filtros, matrículas_selecionadas

class Matrícula:

    def __init__(self, aluno, curso, bolsa, data_início):
        self.aluno = aluno
        self.curso = curso
        self.bolsa = bolsa if bolsa in ('completa', 'parcial', 'nenhuma') else 'nenhuma'
        self.data_início = data_início
        self.nota_final = -1.0

   def __str__(self):
        matrícula_str = 'Matrícula do aluno'
        if self.aluno.sexo == 'F': matrícula_str = 'Matrícula da aluna'
        matrícula_str += ':: ' + str(self.aluno) + '\n    - no curso:: ' + str(self.curso)\
             + '\n    - data início: ' + str(self.data_início);
        if self.bolsa != 'nenhuma': matrícula_str += " - bolsa: " + self.bolsa
        if self.nota_final >= 0: matrícula_str += " - nota final: " + str(self.nota_final)
        return matrícula_str

      def __eq__(self, matrícula):
        if str(self) == str(matrícula): return True
        else: return False
```

Observe que, de forma semelhante ao método __str__ da classe Aluno, as referências aos objetos das classes Aluno, Curso e Data são convertidas pela função str para serem concatenadas como strings.

O método de comparação de igualdade __eq__ foi definido porque será necessário para utilização na função inserir_matrícula que será defnida posteriormente no módulo escola_informática do diretório controle. Quando estivermos comentando a respeito dessa função, será explicada a necessidade da definição do método __eq__ na classe Matrícula.

Na função selecionar_matrículas para o filtro curso_sem_pré_requisitos serão mostradas informações diferenciadas em função de seu valor True ou False, informando curso com ou sem pré-requisitos.

Para obter um atributo de objeto referenciado na classe Matrícula é necessário criar um encadeamento de referências até chegar ao objeto da classe do atributo desejado. Por exemplo, para obtermos o atributo cidade do aluno a partir do objeto de matrícula, temos que obter o objeto aluno e em seguida o objeto endereço, para então obter o atributo cidade, resultando na seguinte expressão: matrícula.aluno.endereço.cidade.

O filtro curso_sem_pré_requisitos é do tipo bool. Para utilizar o comparador de igualdade (==) para comparar diretamente como esse filtro foi criada uma variável a partir da seguinte atribuição: matrícula_curso_sem_pré_requisitos = (len(matrícula.curso.pré_requisitos) == 0). A função len (abreviação de lenght : comprimento no sentido de quantidade de elementos) aplicada ao atributo pré_requisitos do curso da matrícula, retorna quantos pré-requisitos o curso tem. Ao ser comparado com zero, o resultado é True se o curso não tem pré-requisitos e False se o curso tem pelo menos um pré-requisito. Desta forma, a variável matrícula_curso_sem_pré_requisitos também é do tipo bool e pode ser comparada diretamente com o filtro curso_sem_pré_requisitos.

5 - Quarto Passo : Implementando as Funções do Módulo Principal da Aplicação

Para finalizar, vamos definir no diretório controle o módulo escola_informática, utilizado para controlar toda a aplicação.

```
from util.data import Data
from entidades.endereço import Endereço
from entidades.aluno import inserir_aluno, Aluno, get_alunos
from entidades.curso import inserir_curso, Curso, get_cursos
from entidades.matrícula import inserir_matrícula, selecionar_matrículas, get_matrículas

def cadastrar_alunos():
    inserir_aluno(Aluno(nome = 'Ana Julia Parra', rg = '31.845.917', cpf = '212.234.571-32',
       data_nascimento = Data(15, 10, 1982), sexo = 'F',
       endereço = Endereço('Rua Arco Verde', 171, 'apto 301', 'Água Boa', 'Dourados', '79810-015')))
    inserir_aluno(Aluno('Ana Ligia Silveira', '32.870.923', '312.434.775-30', Data(8, 8, 1985), 'F',
        Endereço('Rua Chapéu Velho', 303, None, 'Rouxinol', 'Glória de Dourados', '79820-017')))
    inserir_aluno(Aluno('André Oliveira', '41.825.341', '531.331.740-71', Data(20, 3, 1993), 'M',
        Endereço('Rua Sino da Mata', 303, None, 'Brejão', 'Caarapó', '73100-000')))

def cadastrar_cursos():
    inserir_curso(Curso(título = 'Java Básico', carga_horária_semanal = 8, duração_semanas = 10))
    inserir_curso(Curso('Java Aplicações Locais', 8, 10))
    inserir_curso(Curso('Java Aplicações Web', 8, 10))
    inserir_curso(Curso('Banco de Dados Básico', 8, 8))
    inserir_curso(Curso('Banco de Dados Avançado', 8, 8))
    curso2 = get_cursos()['Java Aplicações Locais']
    curso3 = get_cursos()['Java Aplicações Web']
    curso5 = get_cursos()['Banco de Dados Avançado']
    curso2.inserir_pré_requisito('Java Básico')
    curso2.inserir_pré_requisito('Banco de Dados Básico')
    curso3.inserir_pré_requisito('Java Aplicações Locais')
    curso5.inserir_pré_requisito('Banco de Dados Básico')

def cadastrar_matrículas():
    inserir_matrícula(cpf_aluno = '212.234.571-32', título_curso = 'Java Básico',
       bolsa = 'completa', data_início = Data(1, 3, 2013))
    inserir_matrícula('312.434.775-30', 'Java Básico', 'parcial', Data(1, 3, 2013))
    inserir_matrícula('212.234.571-32', 'Java Aplicações Locais', 'parcial', Data(1, 6, 2013))
    inserir_matrícula('312.434.775-30', 'Java Aplicações Locais', 'nenhuma', Data(1, 6, 2013))
    inserir_matrícula('212.234.571-32', 'Java Aplicações Web', 'parcial', Data(1, 9, 2013))
    inserir_matrícula('312.434.775-30', 'Java Aplicações Web', 'nenhuma', Data(1, 9, 2013))
    inserir_matrícula('212.234.571-32', 'Banco de Dados Básico', 'parcial', Data(1, 3, 2013))
    inserir_matrícula('312.434.775-30', 'Banco de Dados Básico', 'nenhuma', Data(1, 3, 2013))
    inserir_matrícula('531.331.740-71', 'Banco de Dados Básico', 'nenhuma', Data(1, 3, 2013))
    inserir_matrícula('212.234.571-32', 'Banco de Dados Avançado', 'nenhuma', Data(1, 6, 2013))
    inserir_matrícula('531.331.740-71', 'Banco de Dados Avançado', 'nenhuma', Data(1, 6, 2013))
```

```
def imprimir_objetos(cabeçalho, objetos, filtros = None):
    if filtros == None: print('\n' + cabeçalho)
    else: print ('\n' + cabeçalho + filtros)
    for índice, objeto in enumerate(objetos): print(str(índice + 1) + ' - ' + str(objeto))

if __name__ == '__main__':
    cadastrar_alunos()
    imprimir_objetos('--- Alunos cadastrados', get_alunos().values())
    cadastrar_cursos()
    imprimir_objetos('--- Cursos cadastrados', get_cursos().values())
    cadastrar_matrículas()
    imprimir_objetos('--- Matrículas cadastradas', get_matrículas())
    filtros, matrículas_selecionadas = selecionar_matrículas()
    imprimir_objetos('--- Matrículas selecionadas com ', matrículas_selecionadas, filtros)
    filtros, matrículas_selecionadas = selecionar_matrículas(bolsa_matrícula = 'parcial')
    imprimir_objetos('--- Matrículas selecionadas com ', matrículas_selecionadas, filtros)
    filtros, matrículas_selecionadas = selecionar_matrículas('parcial', cidade_aluno = 'Dourados')
    imprimir_objetos('--- Matrículas selecionadas com ', matrículas_selecionadas, filtros)
    filtros, matrículas_selecionadas = selecionar_matrículas('parcial', 'Dourados',
       curso_sem_pré_requisitos = False)
    imprimir_objetos('--- Matrículas selecionadas com ', matrículas_selecionadas, filtros)
```

Agora vamos esclarecer cada parte da implementação do módulo escola_informática. Vamos iniciar pelas importações. Todas as classes e funções definidas em outros módulos precisam ser importadas para poder serem utilizadas neste módulo. Então, é necessário importar todas as classes cujos construtores serão chamados por funções neste módulo (Data, Endereço, Aluno, Curso e Matrícula), bem como a funções globais que serão chamadas neste módulo.

```
from util.data import Data
from entidades.endereço import Endereço
from entidades.aluno import inserir_aluno, Aluno, get_alunos
from entidades.curso import inserir_curso, Curso, get_cursos
from entidades.matrícula import inserir_matrícula, selecionar_matrículas, get_matrículas
```

Neste ponto é importante conceituarmos o conceito de atributo chave para o armazenamento e recuperação de um objeto em um dicionário. Um atributo de uma classe é considerado chave para um conjunto de objetos desta classe, quando seu valor permite identificar um único objeto desta classe. No caso da classe Aluno, observe que o atributo nome não pode ser caracterizado como chave, porque dois alunos podem ter exatamente o mesmo nome, o que não ocorre com o cpf do aluno, porque com um dado valor de cpf é possível identificar uma única pessoa em todo o território nacional.

Mas fique atento, a definição de atributo chave depende da aplicação. Numa escola de informática é perfeitamente aceitável que todos os seus cursos sejam identificados como títulos distintos e, portanto neste caso, o título do curso pode ser utilizado como atributo chave. Observe que isto pode não ser verdade em outras aplicações. Uma universidade pode ter disciplinas com o mesmo título, que neste caso terão que ser identifcadas pela sigla da disciplina, que deverá ser distinta para cada disciplina.

Na classe Matrícula não existe uma única chave, dado que um único objeto desta classe precisa ser identificado por três referências distintas: aluno, curso e data_início. Aparentemente pode parecer que somente aluno e curso seriam necessários para identificar uma única matrícula. No entanto, se um aluno se matricular em um curso, não concluir e posteriormente se matricular de novo no mesmo curso, o atributo data_inicio também passa a ser relevante para identificar uma única matrícula.

Bem, todo esclarecimento a respeito de um único atributo chave foi feito para justificar porque objetos das classes Aluno e Curso foram armazenados em dicionários, indexados por uma única

chave. E também porque objetos da classe Matrícula foram armazenados em uma lista, que não é indexada por uma chave e sim por índices inteiros. A heurística (regra prática) nestes casos é a seguinte: armazenar em dicionários sempre que possível, por que essa estrutura de dados é muito versátil para recuperar um objeto a partir de sua chave, de uma forma muito mais compacta e eficiente do que criar um loop para comparar atributos ou e procurar um dado objeto em uma lista.

Podemos constatar a versatilidade no uso de dicionários nas implementações das funções inserir_aluno e inserir_curso, importadas dos módulos aluno e curso. A função inserir_aluno recebe um objeto aluno como argumento, obtém a chave do aluno (cpf) e somente insere o objeto aluno no dicionário alunos, se a chave ainda não pertencer ao conjunto de chaves do dicionário (obtido com a função keys : chaves). A implementação da função inserir_curso é equivalente.

```
def inserir_aluno(aluno):
    cpf_aluno = aluno.cpf
    if cpf_aluno not in alunos.keys(): alunos[cpf_aluno] = aluno
    else: print('Aluno já cadastrado')
```

No entanto, a função inserir_matrícula, importada do módulo matrícula, é bem diferente: (a) precisa avaliar a existência dos objetos aluno e curso, nos seus respectivos dicionários, a partir de chaves recebidas como argumento; e (b) após criar o objeto da classe Matrícula, precisa avaliar se este objeto já não pertence à lista de matrículas cadastradas, antes de inseri-lo na lista. O teste de não inserção do objeto recém-criado na lista (if matrícula not in matrículas) terá como consequência implícita a necessidade de que o objeto matrícula seja comparado com cada objeto previamente cadastrado na lista matrículas. Por esse motivo, é que na classe Matrícula foi necessária a criação do método __eq__, utilizado para comparar dois objetos da classe Matrícula.

```
def inserir_matrícula(cpf_aluno, título_curso, bolsa, data_início):
    aluno = alunos[cpf_aluno]
    curso = cursos[título_curso]
    if aluno == None:
        print('Matrícula mal sucedida: aluno não cadastrado')
        return
    if curso == None:
        print('Matrícula mal sucedida: curso não cadastrado')
        return
    matrícula = Matrícula(aluno, curso, bolsa, data_início)
    if matrícula == None: pass
    if matrícula not in matrículas: matrículas.append(matrícula)
    else: print('Matrícula já cadastrada --- ' + str(matrícula))
```

Na função cadastrar_alunos, para cada aluno inserido é chamada a função inserir_aluno, importada do módulo aluno, passando como argumento o objeto da classe Aluno, criado a partir do seu construtor. Observe que o construtor da classe Aluno, requer como argumento um objeto da classe Data e um objeto da classe Endereço, ambos criados diretamente a partir da chamada dos construtores de suas respectivas classes.

```
def cadastrar_alunos():
    inserir_aluno(Aluno(nome = 'Ana Julia Parra', rg = '31.845.917', cpf = '212.234.571-32',
       data_nascimento = Data(15, 10, 1982), sexo = 'F',
       endereço = Endereço('Rua Arco Verde', 171, 'apto 301', 'Água Boa', 'Dourados', '79810-015')))
    inserir_aluno(Aluno('Ana Ligia Silveira', '32.870.923', '312.434.775-30', Data(8, 8, 1985), 'F',
        Endereço('Rua Chapéu Velho', 303, None, 'Rouxinol', 'Glória de Dourados', '79820-017')))
    inserir_aluno(Aluno('André Oliveira', '41.825.341', '531.331.740-71', Data(20, 3, 1993), 'M',
        Endereço('Rua Sino da Mata', 303, None, 'Brejão', 'Caarapó', '73100-000')))
```

Na função cadastrar_cursos, utilizamos a função inserir_curso, importada do módulo curso, passando como argumento um objeto da classe Curso, criado a partir da chamada do construtor

dessa classe. Lembre-se que o atributo pré-requisitos não é passado no construtor e, portanto, a função inserir_pré_requisito é utilizada para inserir cada pré-requisito na lista que os armazena no objeto da classe Curso. Inicialmente, são criados os cursos e, em seguida, são obtidos os objetos dos cursos que tem pré-requisitos, para poder chamar a função inserir_pré_requisito da classe Curso e inserir o título de cada pré-requisito.

```
def cadastrar_cursos():
    inserir_curso(Curso(título = 'Java Básico', carga_horária_semanal = 8, duração_semanas = 10))
    inserir_curso(Curso('Java Aplicações Locais', 8, 10))
    inserir_curso(Curso('Java Aplicações Web', 8, 10))
    inserir_curso(Curso('Banco de Dados Básico', 8, 8))
    inserir_curso(Curso('Banco de Dados Avançado', 8, 8))
    curso2 = cursos.get('Java Aplicações Locais')
    curso3 = cursos.get('Java Aplicações Web')
    curso5 = cursos.get('Banco de Dados Avançado')
    curso2.inserir_pré_requisito('Java Básico')
    curso2.inserir_pré_requisito('Banco de Dados Básico')
    curso3.inserir_pré_requisito('Java Aplicações Locais')
    curso5.inserir_pré_requisito('Banco de Dados Básico')
```

Na função cadastrar_matrículas, ocorrem chamadas da função inserir_matrícula, importada do módulo matrícula, que não recebe como argumento um objeto da classe Matrícula, dado que esse objeto será criado diretamente na função inserir_matrícula.

```
def cadastrar_matrículas():
    inserir_matrícula(cpf_aluno = '212.234.571-32', título_curso = 'Java Básico',
       bolsa = 'completa', data_início = Data(1, 3, 2013))
    inserir_matrícula('312.434.775-30', 'Java Básico', 'parcial', Data(1, 3, 2013))
    inserir_matrícula('212.234.571-32', 'Java Aplicações Locais', 'parcial', Data(1, 6, 2013))
    inserir_matrícula('312.434.775-30', 'Java Aplicações Locais', 'nenhuma', Data(1, 6, 2013))
    inserir_matrícula('212.234.571-32', 'Java Aplicações Web', 'parcial', Data(1, 9, 2013))
    inserir_matrícula('312.434.775-30', 'Java Aplicações Web', 'nenhuma', Data(1, 9, 2013))
    inserir_matrícula('212.234.571-32', 'Banco de Dados Básico', 'parcial', Data(1, 3, 2013))
    inserir_matrícula('312.434.775-30', 'Banco de Dados Básico', 'nenhuma', Data(1, 3, 2013))
    inserir_matrícula('531.331.740-71', 'Banco de Dados Básico', 'nenhuma', Data(1, 3, 2013))
    inserir_matrícula('212.234.571-32', 'Banco de Dados Avançado', 'nenhuma', Data(1, 6, 2013))
    inserir_matrícula('531.331.740-71', 'Banco de Dados Avançado', 'nenhuma', Data(1, 6, 2013))
```

A função imprimir_objetos já foi ilustrada no Tutorial 2.

```
def imprimir_objetos(cabeçalho, objetos, filtros = None):
    if filtros == None: print('\n' + cabeçalho)
    else: print ('\n' + cabeçalho + filtros)
    for índice, objeto in enumerate(objetos): print(str(índice + 1) + ' - ' + str(objeto))
```

Finalmente, será executado o bloco principal da aplicação. Inicialmente são cadastrados e impressos os alunos, os cursos e as matrículas. Em seguida, a função selecionar_matrículas, importada do módulo matrícula, é chamada com várias configurações de filtros para retornar o string com os filtros utilizados e a lista de matrículas selecionadas, que serão utilizados para impressão. Aqui utilizamos a mesma estratégia de escolha de filtros definida no Tutorial 2. Inicialmente, sem nenhum filtro obrigatório a lista de selecionados é equivalente à lista de cadastrados e a medida que cada filtro adicional é tornado obrigatório, a lista de selecionados diminui pelo menos de um objeto da classe Matrícula. Observe que somente os objetos da classe Matrícula são selecionados, mas que a seleção de cada um deles vai ser testada utilizando atributos obtidos a partir das classes referenciadas pelos objetos da classe Matrícula, conforme explicado anteriormente ao comentarmos o método selecionar_matrículas.

```
if __name__ == '__main__':
    cadastrar_alunos()
    imprimir_objetos('--- Alunos cadastrados', alunos.values())
```

```
    cadastrar_cursos()
    imprimir_objetos('--- Cursos cadastrados', cursos.values())
    cadastrar_matrículas()
    imprimir_objetos('--- Matrículas cadastradas', matrículas)
    filtros, matrículas_selecionadas = selecionar_matrículas(matrículas)
    imprimir_objetos('--- Matrículas selecionadas com ', matrículas_selecionadas, filtros)
    filtros, matrículas_selecionadas = selecionar_matrículas(matrículas,
       bolsa_matrícula = 'parcial')
    imprimir_objetos('--- Matrículas selecionadas com ', matrículas_selecionadas, filtros)
    filtros, matrículas_selecionadas = selecionar_matrículas(matrículas, 'parcial',
       cidade_aluno = 'Dourados')
    imprimir_objetos('--- Matrículas selecionadas com ', matrículas_selecionadas, filtros)
    filtros, matrículas_selecionadas = selecionar_matrículas(matrículas, 'parcial', 'Dourados',
       curso_sem_pré_requisitos = False)
    imprimir_objetos('--- Matrículas selecionadas com ', matrículas_selecionadas, filtros)
```

A saída da execução da aplicação é a seguinte:

```
Alunos cadastrados
1 - Ana Julia Parra - RG:31.845.917 - CPF:212.234.571-32 - nascimento:15/10/1982 - sexo:feminino
    - residente em:: Rua Arco Verde - 171 - apto 301 - bairro: Água Boa - Dourados - CEP: 79810-015
2 - Ana Ligia Silveira - RG:32.870.923 - CPF:312.434.775-30 - nascimento:08/08/1985 - sexo:feminino
    - residente em:: Rua Chapéu Velho - 303 - bairro: Rouxinol - Glória de Dourados - CEP: 79820-017
3 - André Oliveira - RG:41.825.341 - CPF:531.331.740-71 - nascimento:20/03/1993 - sexo:masculino
    - residente em:: Rua Sino da Mata - 303 - bairro: Brejão - Caarapó - CEP: 73100-000


Cursos cadastrados
1 - Java Básico -- carga horária semanal: 8 - duração em semanas: 10
2 - Java Aplicações Locais -- carga horária semanal: 8 - duração em semanas: 10
    - pré-requisitos: Java Básico - Banco de Dados Básico
3 - Java Aplicações Web -- carga horária semanal: 8 - duração em semanas: 10
    - pré-requisitos: Java Aplicações Locais
4 - Banco de Dados Básico -- carga horária semanal: 8 - duração em semanas: 8
5 - Banco de Dados Avançado -- carga horária semanal: 8 - duração em semanas: 8
    - pré-requisitos: Banco de Dados Básico




Matrículas cadastradas
1 - Matrícula da aluna
    Ana Julia Parra - RG:31.845.917 - CPF:212.234.571-32 - nascimento:15/10/1982 - sexo:feminino
    - residente em:: Rua Arco Verde - 171 - apto 301 - bairro: Água Boa - Dourados - CEP: 79810-015
    - no curso:: Java Básico -- carga horária semanal: 8 - duração em semanas: 10
    - data início: 01/03/2013 - bolsa: completa
2 - Matrícula da aluna
    Ana Ligia Silveira - RG:32.870.923 - CPF:312.434.775-30 - nascimento:08/08/1985 - sexo:feminino
    - residente em:: Rua Chapéu Velho - 303 - bairro: Rouxinol - Glória de Dourados - CEP: 79820-017
    - no curso:: Java Básico -- carga horária semanal: 8 - duração em semanas: 10
    - data início: 01/03/2013 - bolsa: parcial
3 - Matrícula da aluna
    Ana Julia Parra - RG:31.845.917 - CPF:212.234.571-32 - nascimento:15/10/1982 - sexo:feminino
    - residente em:: Rua Arco Verde - 171 - apto 301 - bairro: Água Boa - Dourados - CEP: 79810-015
    - no curso:: Java Aplicações Locais -- carga horária semanal: 8 - duração em semanas: 10
    - pré-requisitos: Java Básico - Banco de Dados Básico
    - data início: 01/06/2013 - bolsa: parcial
4 - Matrícula da aluna
    Ana Ligia Silveira - RG:32.870.923 - CPF:312.434.775-30 - nascimento:08/08/1985 - sexo:feminino
    - residente em:: Rua Chapéu Velho - 303 - bairro: Rouxinol - Glória de Dourados - CEP: 79820-017
    - no curso:: Java Aplicações Locais -- carga horária semanal: 8 - duração em semanas: 10
    - pré-requisitos: Java Básico - Banco de Dados Básico
    - data início: 01/06/2013
5 - Matrícula da aluna
    Ana Julia Parra - RG:31.845.917 - CPF:212.234.571-32 - nascimento:15/10/1982 - sexo:feminino
    - residente em:: Rua Arco Verde - 171 - apto 301 - bairro: Água Boa - Dourados - CEP: 79810-015
    - no curso:: Java Aplicações Web -- carga horária semanal: 8 - duração em semanas: 10
    - pré-requisitos: Java Aplicações Locais
    - data início: 01/09/2013 - bolsa: parcial
6 - Matrícula da aluna
    Ana Ligia Silveira - RG:32.870.923 - CPF:312.434.775-30 - nascimento:08/08/1985 - sexo:feminino
    - residente em:: Rua Chapéu Velho - 303 - bairro: Rouxinol - Glória de Dourados - CEP: 79820-017
```

```
    - no curso:: Java Aplicações Web -- carga horária semanal: 8 - duração em semanas: 10
    - pré-requisitos: Java Aplicações Locais
    - data início: 01/09/2013
7 - Matrícula da aluna
    Ana Julia Parra - RG:31.845.917 - CPF:212.234.571-32 - nascimento:15/10/1982 - sexo:feminino
    - residente em:: Rua Arco Verde - 171 - apto 301 - bairro: Água Boa - Dourados - CEP: 79810-015
    - no curso:: Banco de Dados Básico -- carga horária semanal: 8 - duração em semanas: 8
    - data início: 01/03/2013 - bolsa: parcial
8 - Matrícula da aluna
    Ana Ligia Silveira - RG:32.870.923 - CPF:312.434.775-30 - nascimento:08/08/1985 - sexo:feminino
    - residente em:: Rua Chapéu Velho - 303 - bairro: Rouxinol - Glória de Dourados - CEP: 79820-017
    - no curso:: Banco de Dados Básico -- carga horária semanal: 8 - duração em semanas: 8
    - data início: 01/03/2013
9 - Matrícula do aluno
    André Oliveira - RG:41.825.341 - CPF:531.331.740-71 - nascimento:20/03/1993 - sexo:masculino
    - residente em:: Rua Sino da Mata - 303 - bairro: Brejão - Caarapó - CEP: 73100-000
    - no curso:: Banco de Dados Básico -- carga horária semanal: 8 - duração em semanas: 8
    - data início: 01/03/2013
10 - Matrícula da aluna
    Ana Julia Parra - RG:31.845.917 - CPF:212.234.571-32 - nascimento:15/10/1982 - sexo:feminino
    - residente em:: Rua Arco Verde - 171 - apto 301 - bairro: Água Boa - Dourados - CEP: 79810-015
    - no curso:: Banco de Dados Avançado -- carga horária semanal: 8 - duração em semanas: 8
    - pré-requisitos: Banco de Dados Básico
    - data início: 01/06/2013
11 - Matrícula do aluno
    André Oliveira - RG:41.825.341 - CPF:531.331.740-71 - nascimento:20/03/1993 - sexo:masculino
    - residente em:: Rua Sino da Mata - 303 - bairro: Brejão - Caarapó - CEP: 73100-000
    - no curso:: Banco de Dados Avançado -- carga horária semanal: 8 - duração em semanas: 8
    - pré-requisitos: Banco de Dados Básico
    - data início: 01/06/2013


Matrículas selecionadas com Filtros:
1 - Matrícula da aluna
    Ana Julia Parra - RG:31.845.917 - CPF:212.234.571-32 - nascimento:15/10/1982 - sexo:feminino
    - residente em:: Rua Arco Verde - 171 - apto 301 - bairro: Água Boa - Dourados - CEP: 79810-015
    - no curso:: Java Básico -- carga horária semanal: 8 - duração em semanas: 10
    - data início: 01/03/2013 - bolsa: completa
2 - Matrícula da aluna
    Ana Ligia Silveira - RG:32.870.923 - CPF:312.434.775-30 - nascimento:08/08/1985 - sexo:feminino
    - residente em:: Rua Chapéu Velho - 303 - bairro: Rouxinol - Glória de Dourados - CEP: 79820-017
    - no curso:: Java Básico -- carga horária semanal: 8 - duração em semanas: 10
    - data início: 01/03/2013 - bolsa: parcial
3 - Matrícula da aluna
    Ana Julia Parra - RG:31.845.917 - CPF:212.234.571-32 - nascimento:15/10/1982 - sexo:feminino
    - residente em:: Rua Arco Verde - 171 - apto 301 - bairro: Água Boa - Dourados - CEP: 79810-015
    - no curso:: Java Aplicações Locais -- carga horária semanal: 8 - duração em semanas: 10
    - pré-requisitos: Java Básico - Banco de Dados Básico
    - data início: 01/06/2013 - bolsa: parcial
4 - Matrícula da aluna
    Ana Ligia Silveira - RG:32.870.923 - CPF:312.434.775-30 - nascimento:08/08/1985 - sexo:feminino
    - residente em:: Rua Chapéu Velho - 303 - bairro: Rouxinol - Glória de Dourados - CEP: 79820-017
    - no curso:: Java Aplicações Locais -- carga horária semanal: 8 - duração em semanas: 10
    - pré-requisitos: Java Básico - Banco de Dados Básico
    - data início: 01/06/2013
5 - Matrícula da aluna
    Ana Julia Parra - RG:31.845.917 - CPF:212.234.571-32 - nascimento:15/10/1982 - sexo:feminino
    - residente em:: Rua Arco Verde - 171 - apto 301 - bairro: Água Boa - Dourados - CEP: 79810-015
    - no curso:: Java Aplicações Web -- carga horária semanal: 8 - duração em semanas: 10
    - pré-requisitos: Java Aplicações Locais
    - data início: 01/09/2013 - bolsa: parcial
6 - Matrícula da aluna
    Ana Ligia Silveira - RG:32.870.923 - CPF:312.434.775-30 - nascimento:08/08/1985 - sexo:feminino
    - residente em:: Rua Chapéu Velho - 303 - bairro: Rouxinol - Glória de Dourados - CEP: 79820-017
    - no curso:: Java Aplicações Web -- carga horária semanal: 8 - duração em semanas: 10
    - pré-requisitos: Java Aplicações Locais
    - data início: 01/09/2013
7 - Matrícula da aluna
    Ana Julia Parra - RG:31.845.917 - CPF:212.234.571-32 - nascimento:15/10/1982 - sexo:feminino
    - residente em:: Rua Arco Verde - 171 - apto 301 - bairro: Água Boa - Dourados - CEP: 79810-015
    - no curso:: Banco de Dados Básico -- carga horária semanal: 8 - duração em semanas: 8
    - data início: 01/03/2013 - bolsa: parcial
8 - Matrícula da aluna
    Ana Ligia Silveira - RG:32.870.923 - CPF:312.434.775-30 - nascimento:08/08/1985 - sexo:feminino
    - residente em:: Rua Chapéu Velho - 303 - bairro: Rouxinol - Glória de Dourados - CEP: 79820-017
```

```
    - no curso:: Banco de Dados Básico -- carga horária semanal: 8 - duração em semanas: 8
    - data início: 01/03/2013
9 - Matrícula do aluno
    André Oliveira - RG:41.825.341 - CPF:531.331.740-71 - nascimento:20/03/1993 - sexo:masculino
    - residente em:: Rua Sino da Mata - 303 - bairro: Brejão - Caarapó - CEP: 73100-000
    - no curso:: Banco de Dados Básico -- carga horária semanal: 8 - duração em semanas: 8
    - data início: 01/03/2013
10 - Matrícula da aluna
    Ana Julia Parra - RG:31.845.917 - CPF:212.234.571-32 - nascimento:15/10/1982 - sexo:feminino
    - residente em:: Rua Arco Verde - 171 - apto 301 - bairro: Água Boa - Dourados - CEP: 79810-015
    - no curso:: Banco de Dados Avançado -- carga horária semanal: 8 - duração em semanas: 8
    - pré-requisitos: Banco de Dados Básico
    - data início: 01/06/2013
11 - Matrícula do aluno
    André Oliveira - RG:41.825.341 - CPF:531.331.740-71 - nascimento:20/03/1993 - sexo:masculino
    - residente em:: Rua Sino da Mata - 303 - bairro: Brejão - Caarapó - CEP: 73100-000
    - no curso:: Banco de Dados Avançado -- carga horária semanal: 8 - duração em semanas: 8
    - pré-requisitos: Banco de Dados Básico
    - data início: 01/06/2013


Matrículas selecionadas com Filtros: bolsa da matrícula: parcial
1 - Matrícula da aluna
    Ana Ligia Silveira - RG:32.870.923 - CPF:312.434.775-30 - nascimento:08/08/1985 - sexo:feminino
    - residente em:: Rua Chapéu Velho - 303 - bairro: Rouxinol - Glória de Dourados - CEP: 79820-017
    - no curso:: Java Básico -- carga horária semanal: 8 - duração em semanas: 10
    - data início: 01/03/2013 - bolsa: parcial
2 - Matrícula da aluna
    Ana Julia Parra - RG:31.845.917 - CPF:212.234.571-32 - nascimento:15/10/1982 - sexo:feminino
    - residente em:: Rua Arco Verde - 171 - apto 301 - bairro: Água Boa - Dourados - CEP: 79810-015
    - no curso:: Java Aplicações Locais -- carga horária semanal: 8 - duração em semanas: 10
    - pré-requisitos: Java Básico - Banco de Dados Básico
    - data início: 01/06/2013 - bolsa: parcial
3 - Matrícula da aluna
    Ana Julia Parra - RG:31.845.917 - CPF:212.234.571-32 - nascimento:15/10/1982 - sexo:feminino
    - residente em:: Rua Arco Verde - 171 - apto 301 - bairro: Água Boa - Dourados - CEP: 79810-015
    - no curso:: Java Aplicações Web -- carga horária semanal: 8 - duração em semanas: 10
    - pré-requisitos: Java Aplicações Locais
    - data início: 01/09/2013 - bolsa: parcial
4 - Matrícula da aluna
    Ana Julia Parra - RG:31.845.917 - CPF:212.234.571-32 - nascimento:15/10/1982 - sexo:feminino
    - residente em:: Rua Arco Verde - 171 - apto 301 - bairro: Água Boa - Dourados - CEP: 79810-015
    - no curso:: Banco de Dados Básico -- carga horária semanal: 8 - duração em semanas: 8
    - data início: 01/03/2013 - bolsa: parcial


Matrículas selecionadas com Filtros: bolsa da matrícula: parcial - cidade do aluno: Dourados
1 - Matrícula da aluna
    Ana Julia Parra - RG:31.845.917 - CPF:212.234.571-32 - nascimento:15/10/1982 - sexo:feminino
    - residente em:: Rua Arco Verde - 171 - apto 301 - bairro: Água Boa - Dourados - CEP: 79810-015
    - no curso:: Java Aplicações Locais -- carga horária semanal: 8 - duração em semanas: 10
    - pré-requisitos: Java Básico - Banco de Dados Básico
    - data início: 01/06/2013 - bolsa: parcial
2 - Matrícula da aluna
    Ana Julia Parra - RG:31.845.917 - CPF:212.234.571-32 - nascimento:15/10/1982 - sexo:feminino
    - residente em:: Rua Arco Verde - 171 - apto 301 - bairro: Água Boa - Dourados - CEP: 79810-015
    - no curso:: Java Aplicações Web -- carga horária semanal: 8 - duração em semanas: 10
    - pré-requisitos: Java Aplicações Locais
    - data início: 01/09/2013 - bolsa: parcial
3 - Matrícula da aluna
    Ana Julia Parra - RG:31.845.917 - CPF:212.234.571-32 - nascimento:15/10/1982 - sexo:feminino
    - residente em:: Rua Arco Verde - 171 - apto 301 - bairro: Água Boa - Dourados - CEP: 79810-015
    - no curso:: Banco de Dados Básico -- carga horária semanal: 8 - duração em semanas: 8
    - data início: 01/03/2013 - bolsa: parcial
```

```
Matrículas selecionadas com Filtros: bolsa da matrícula: parcial - cidade do aluno: Dourados
    - curso com pré-requisitos
1 - Matrícula da aluna
    Ana Julia Parra - RG:31.845.917 - CPF:212.234.571-32 - nascimento:15/10/1982 - sexo:feminino
    - residente em:: Rua Arco Verde - 171 - apto 301 - bairro: Água Boa - Dourados - CEP: 79810-015
    - no curso:: Java Aplicações Locais -- carga horária semanal: 8 - duração em semanas: 10
    - pré-requisitos: Java Básico - Banco de Dados Básico
    - data início: 01/06/2013 - bolsa: parcial
2 - Matrícula da aluna
    Ana Julia Parra - RG:31.845.917 - CPF:212.234.571-32 - nascimento:15/10/1982 - sexo:feminino
    - residente em:: Rua Arco Verde - 171 - apto 301 - bairro: Água Boa - Dourados - CEP: 79810-015
    - no curso:: Java Aplicações Web -- carga horária semanal: 8 - duração em semanas: 10
    - pré-requisitos: Java Aplicações Locais
    - data início: 01/09/2013 - bolsa: parcial
```

Você deve ter percebido, que as notas finais não foram atribuídas e nem os pré-requisitos foram checados. Vamos deixar essa alteração na aplicação como exercício.

Exercícios

Exercício 1 – Incremente a aplicação Escola de Informática, para atribuir notas finais às matrículas cadastradas previamente e verificar os pré-requisitos por ocasião do cadastro da matrícula:
a) substitua a função cadastrar_matrículas por três funções (com sufixos: 1, 2 e 3) para cadastrar as matrículas na mesma data: março, junho e setembro;
b) implemente três funções atribuir_notas (com sufixos: 1, 2 e 3) no módulo escola_informática, atribuindo notas aprovadas (>= 7) e reprovadas;
c) execute o respectivo atribuir_notas, com as notas associadas às disciplinas concluídas, após cada cadastro de matrícula;
d) altere o método inserir_matrícula para aceitar somente matrículas de alunos que tenham sido aprovados nos pré-requisitos.

Exercício 2 – Implemente a aplicação venda de produtos:
a) cadastrando objetos das seguintes classes entidade: Venda (cliente, produto, data, entrega rápida [sim, não]), Cliente (nome, cpf, cnpj, email, endereço) e Produto (descrição, tipo [eletrodoméstico, eletrônico, roupa]);
b) e selecionando as vendas cadastradas com os seguintes filtros: entrega rápida da venda, prefixo do nome do cliente e tipo do produto.

Exercício 3 – Implemente uma aplicação de sua escolha atendendo os seguintes requisitos:
a) implementando três classe entidades, utilizando pelo menos: um atributo do tipo enumerado e um atributo do tipo bool, em pelo menos uma das classes;
b) e definindo os filtros que julgar necessário, incluindo pelo menos um filtro baseado em um atributo enumerado e o outro baseado em um atributo de tipo bool.