Disciplina: Linguagem de Programação II
Professor: Raimundo Osvaldo Vieira

Estudo de Caso 03

Sistema de Controle de Bibliotecas

Esse estudo de caso foi proposto originalmente pela professora Cecília Rubira (UNICAMP)

Neste estudo de caso vamos desenvolver uma pequena aplicação onde são empregados os quatro conceitos fundamentais de orientação a objetos: ***abstração, encapsulamento, herança e polimorfismo****. Conheceremos também mais algumas técnicas de modelagem especialmente úteis para programação orientada a objetos. Como tarefa final, abordaremos o conceito de* ***classe abstrata****.*

1. Descrição do Problema

A aplicação que vamos desenvolver se destina a uma escola que possui uma biblioteca aberta aos seus alunos, professores e ao público em geral. O objetivo do sistema é manter um registro dos empréstimos efetuados, visando controlar a situação de cada volume individualmente e garantir que os empréstimos sejam efetuados de acordo com as normas da biblioteca, descritas a seguir.

Os livros só podem ser retirados da biblioteca por usuários cadastrados numa das seguintes categorias: aluno da escola, professor ou usuário externo. Os alunos devem renovar seu cadastro a cada período letivo.

O número máximo de volumes que um usuário pode retirar, num mesmo período, e o prazo de empréstimo dependem da categoria do usuário, de acordo com a seguinte tabela:

Categoria	Quantidade	Dias de Prazo
Usuários comuns	2	4
Alunos	3	7
Professores	5	14

Os limites acima são reduzidos nos seguintes casos:

(i) o aluno com cadastro vencido fica sujeito aos mesmos limites de um usuário comum, até que providencie sua renovação;

(ii) o usuário com algum prazo de devolução vencido fica impedido de retirar outros volumes, retornando à sua condição normal após a devolução do(s) livro(s) em atraso;

(iii) periódicos, como revistas e jornais, só podem ser retirados por professores, por um prazo máximo de 7 dias;

(iv) um professor pode bloquear um número qualquer de livros ou periódicos, impedindo que os mesmos sejam retirados da biblioteca durante um período de até 20 dias.

2. Projeto das Classes

Uma primeira análise das informações acima nos permite perceber a existência de dois principais tipos de objetos envolvidos no problema: de um lado os **usuários** cadastrados e de outro os **livros** do acervo da biblioteca. As principais operações envolvidas são **empréstimos** e **devoluções** de livros pelos usuários.

Os usuários são divididos em três categorias: usuários externos, alunos e professores. Os usuários externos são os mais limitados quanto às operações que podem efetuar. Iremos considerar alunos e professores como subtipos desses usuários, já que tanto um aluno como um professor pode "substituir" um usuário em qualquer das operações previstas para o mesmo: retirada e devolução de livros. Note que o inverso não é verdadeiro: as operações de bloqueio de livros e renovação de cadastro são exclusivas de professores e alunos, respectivamente.

As normas para empréstimo diferenciam periódicos dos livros em geral, embora sejam aceitas as mesmas operações para ambos. Existem apenas restrições extras para o empréstimo de periódicos. Temos, portanto, ao menos três alternativas para modelar esses objetos:

(a) englobar livros e periódicos num mesmo tipo que tem a forma de publicação (livro comum ou periódico) como um de seus atributos, a ser utilizado nas operações de empréstimo;

(b) considerar periódicos como subtipo de livros, que tem uma operação de empréstimo especializada;

(c) o inverso da opção (b), considerando livros como subtipo de periódicos.

A maior vantagem da opção (a) é reduzir o número de tipos utilizados no projeto, embora implique numa maior complexidade na definição e implementação das operações. As opções (b) e (c) são equivalentes, do ponto de vista da complexidade do projeto e implementação. Iremos optar pela opção (b), por nos parecer mais natural.

3. Diagramas de Classes

Para implementar as hierarquias de tipos acima especificadas iremos definir as seguintes classes para a aplicação:

`Usuario`

Abrange todos os usuários externos e é a superclasse de alunos e professores.
Atributos: nome do usuário e a lista dos livros retirados.
Operações: registro de novo usuário, empréstimo e devolução.

`UsuarioAluno`

Abrange os alunos que estão cadastrados como usuários da biblioteca. É uma subclasse de `Usuario`.
Atributos especializados: data de expiração do cadastro.
Operações especializadas: renovação do cadastro.

`UsuarioProfessor`

Abrange os professores que estão cadastrados como usuários da biblioteca. É uma subclasse de `Usuario`.
Atributos especializados: não tem.
Operações especializadas: bloqueio e desbloqueio de livros.

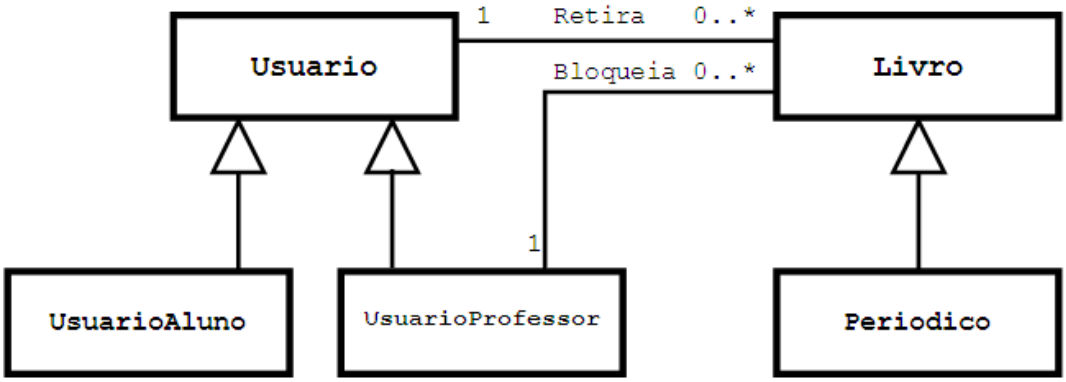
`Livro`

Abrange os livros pertencentes ao acervo da biblioteca.
Atributos: título, dados do empréstimo (usuário, data do empréstimo e data de devolução prevista), caso esteja emprestado, e dados do bloqueio (professor, data do bloqueio e data para desbloqueio), caso esteja bloqueado.
Operações: empréstimo, retorno, bloqueio e desbloqueio.

`Periodico`

Abrange os periódicos que pertencem ao acervo da biblioteca. É uma subclasse de `Livro`.
Atributos especializados: não tem.
Operações especializadas: empréstimo.

A figura abaixo ilustra a hierarquia de classes correspondente.



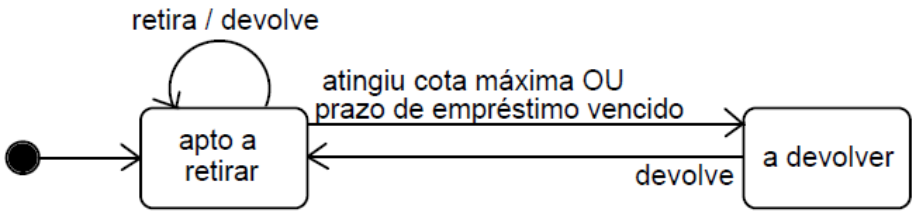
Note que, além dos relacionamentos de herança, estão representados no diagrama dois outros relacionamentos: **Retira** (um usuário pode retirar livros) e **Bloqueia** (um professor pode bloquear livros). Na notação UML, esse tipo de relacionamento onde há uma conexão física, lógica ou conceitual entre duas classes de objetos, é denominado **associação**.

4. Diagramas de Estados

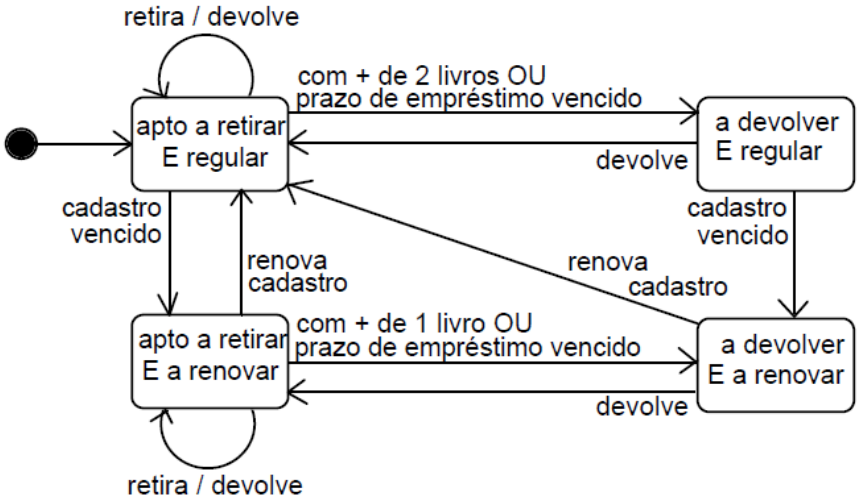
No estudo de caso 01 (Terminal de Caixa Automático) aprendemos a construir diagramas de sequência para ilustrar a execução dos métodos. Agora, vamos aprender a representar os estados e as mudanças de estado dos objetos.

Segundo as normas da biblioteca, a ocorrência de alguns eventos implica em mudanças no estado de um determinado usuário. São eles: a expiração do prazo de validade do cadastro de um aluno, que passa a ser tratado como um usuário normal, e o vencimento de um prazo de empréstimo, que impede novos empréstimos pelo usuário. Dizemos que um usuário qualquer tem dois estados possíveis: apto a retirar livros ou impedido de efetuar novos empréstimos. Um aluno poderá ainda estar em dois "super estados": com cadastro regular ou com cadastro vencido, que se combinam com os dois estados possíveis para um usuário qualquer, resultando em quatro estados possíveis para uma aluno: com cadastro regular e apto a retirar livros, com cadastro regular e impedido de efetuar novos empréstimos, com cadastro vencido e apto a retirar livros e, finalmente, com cadastro vencido e impedido de efetuar novos empréstimos.

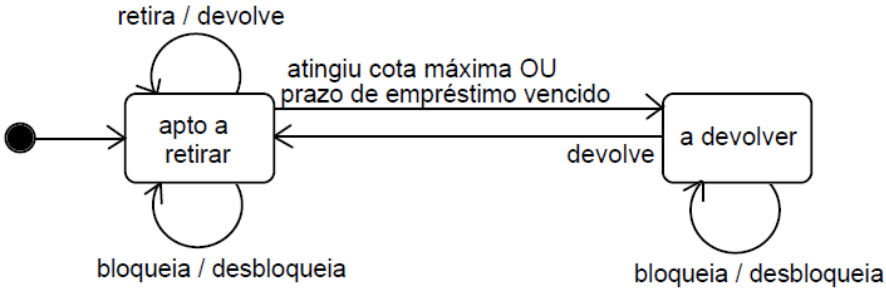
A figura abaixo mostra as mudanças de estado possíveis para um usuário externo.



A figura seguinte ilustra as mudanças de estado possíveis para um aluno. Note que, na figura anterior, o único tipo de operação permitida no estado "a devolver" é a devolução de um livro. Os eventos "atingiu cota máxima" e "prazo de empréstimo vencido" são disparados automaticamente pelo sistema. A retirada de um livro, por exemplo, pode implicar em ser atingida a cota máxima para aquele usuário.

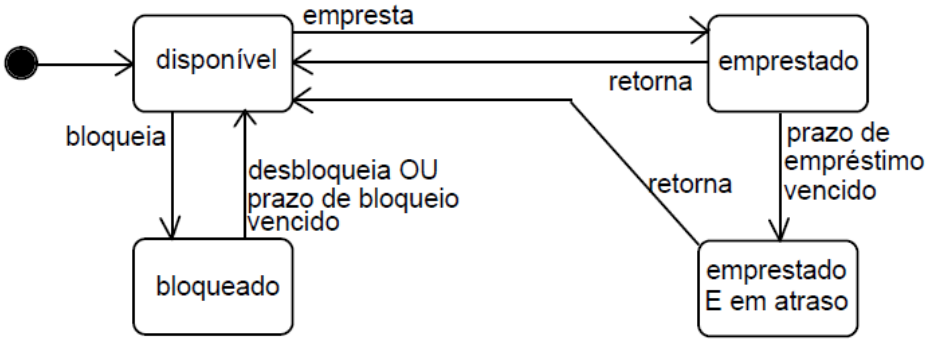


O comportamento de um professor é semelhante ao de um usuário externo, admitindo-se porém as operações de bloqueio e desbloqueio, conforme representado na figura a seguir.



De acordo com o diagrama, as operações de bloqueio e desbloqueio são permitidas mesmo que o professor esteja com algum prazo de empréstimo vencido, embora na análise do problema não tenha sido feita qualquer menção explícita a esse respeito. A elaboração desses diagramas é de grande auxílio na detecção de eventuais omissões e inconsistências nas especificações iniciais do sistema.

Um livro está, também, sujeito a mudanças de comportamento em função de eventos como empréstimos, devoluções, bloqueios e desbloqueios. A figura abaixo apresenta o diagrama de estados correspondente.



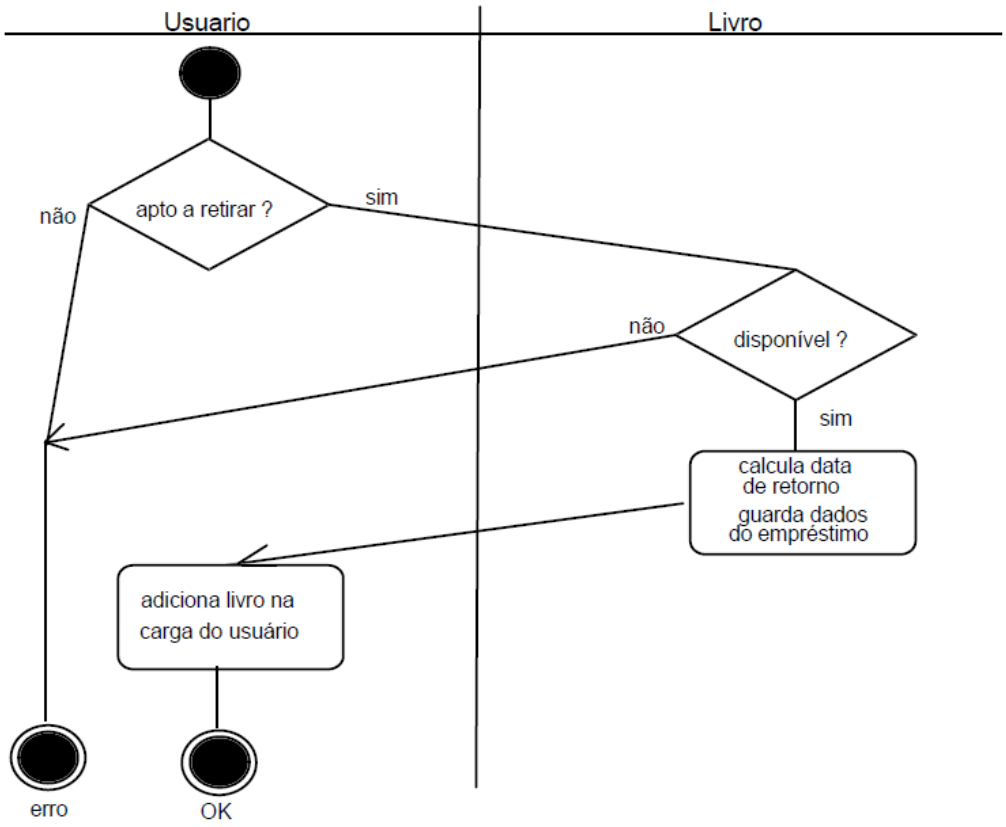
Observe que, de acordo com o diagrama, um livro emprestado não pode ser bloqueado e o desbloqueio de um livro bloqueado ocorre automaticamente ao final do prazo ou por uma ordem de desbloqueio. Estes são outros exemplos de decisões de projeto que os diagramas permitem destacar.

5. Diagramas de Atividades

Nos diagramas anteriores estão representados os relacionamentos entre as diferentes classes (diagrama de classes) e a dinâmica dos objetos de cada classe isoladamente (diagramas de estados).

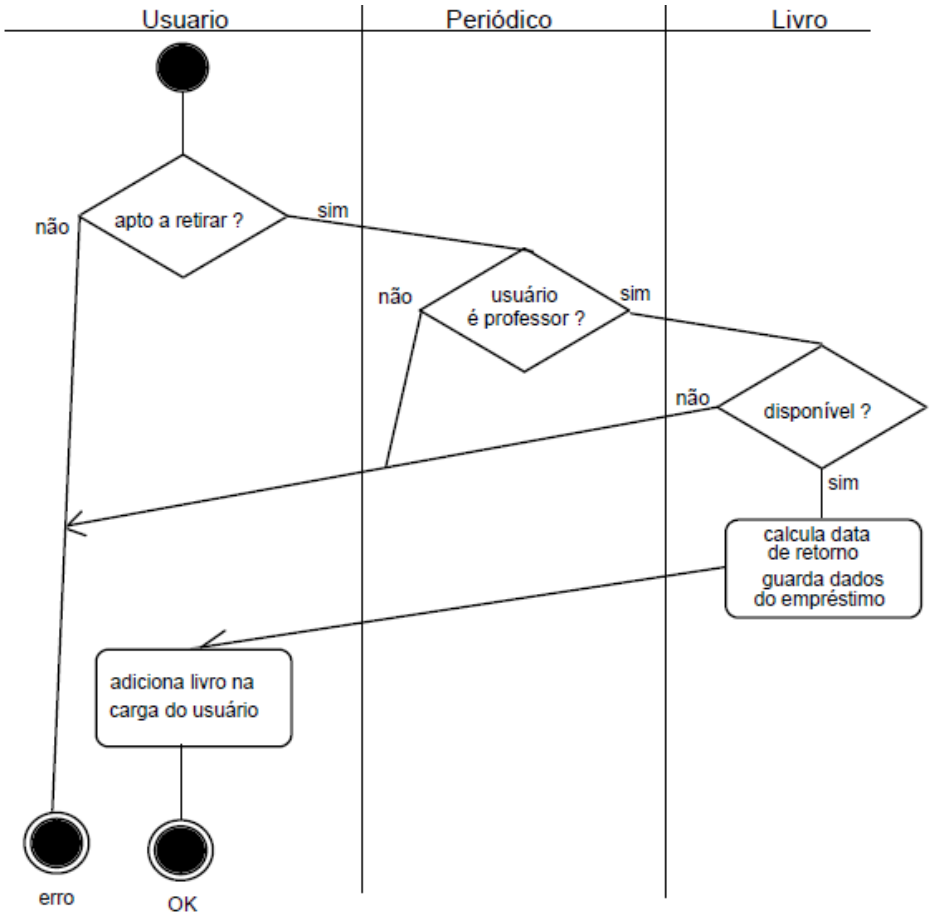
Um terceiro aspecto importante a ser modelado é a interação entre os objetos para a realização de cada ação do sistema, de forma a definir as responsabilidades de cada um. Para isso

utilizaremos diagramas de atividades. Abaixo, é mostrado o diagrama de atividades para a operação **empréstimo de livro**.

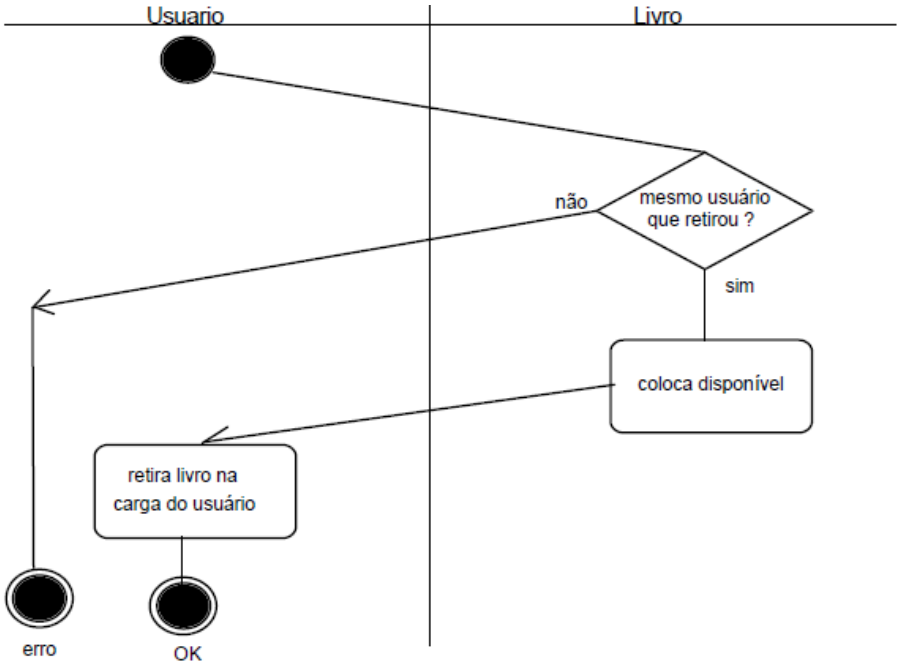


Observe que um diagrama de atividades assemelha-se a um diagrama de seqüência. Ambos retratam a colaboração entre os objetos a fim de realizar uma operação. O diagrama de atividades, porém, o faz de forma mais detalhada.

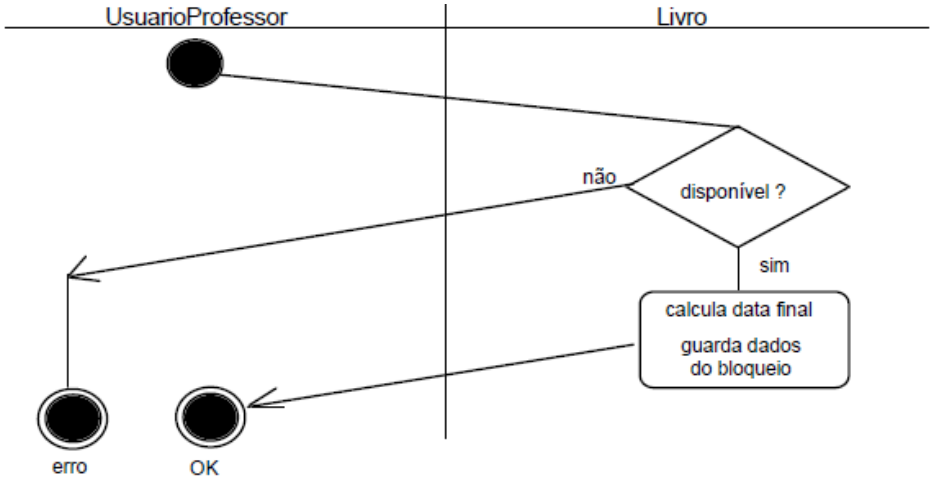
A seguir, é apresentado o diagrama de atividades para a operação **empréstimo de periódico**.



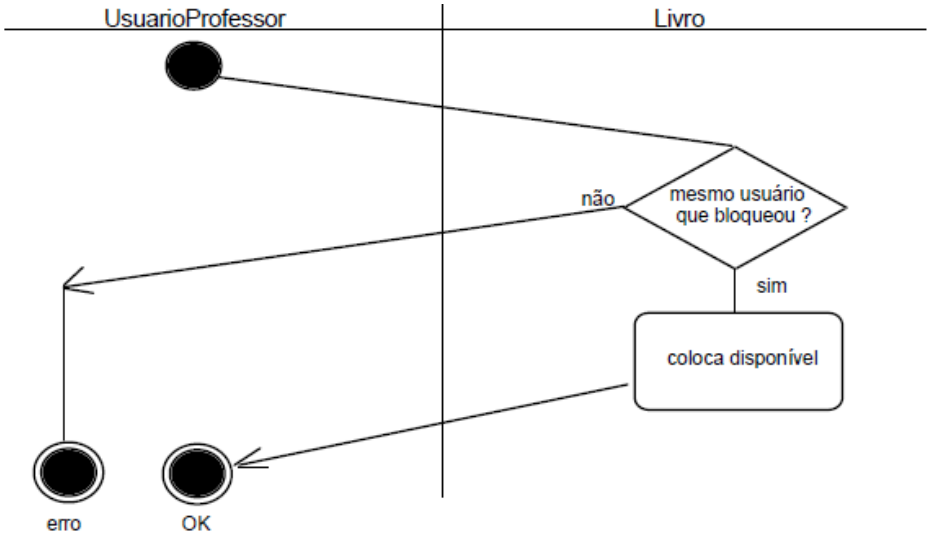
O diagrama de atividades para a operação **devolução** é ilustrado abaixo.



Na sequência é mostrado o diagrama de atividades para a operação **bloqueio**.



Abaixo é mostrado o diagrama para **desbloqueio**.



Com base nesses diagramas, a implementação dos métodos correspondentes às operações ficará mais fácil.

6. Implementação das Classes

6.1. Classe `Usuario`

No projeto da classe `Usuario` estão definidos como atributos o nome do usuário e a lista de livros retirados. O nome do usuário pode ser armazenado num atributo do tipo `String`, que é uma classe nativa de Java para representar cadeias de caracteres. Para armazenar a lista de livros retirados, iremos utilizar a classe `ArrayList`.

As operações definidas no projeto da classe são: criação de novo usuário, retirada de livro e devolução de livro. A criação de um novo usuário pode ser implementada através de um método construtor que receba como parâmetro o nome do usuário. Para as outras operações definiremos dois métodos: `retiraLivro()` e `devolveLivro()`. Ambos recebem como parâmetro o livro que está sendo retirado ou devolvido e retornam uma condição para indicar se a operação foi aceita.

A implementação da classe tem a seguinte estrutura geral:

```
import java.util.ArrayList;
import java.util.Iterator;

public class Usuario {

      private String nome;
      private ArrayList<Livro> livrosRetirados;

      public Usuario (String st) {
         this.nome=st;
         this.livrosRetirados = new ArrayList<Livro>();
      }

      public boolean retiraLivro (Livro it) {

      }

      public boolean devolveLivro (Livro it) {

      }
}
```

O construtor simplesmente guarda o nome do usuário e cria um ArrayList para guardar a lista de livros retirados.

Além dos métodos já ilustrados, vamos precisar de alguns métodos auxiliares para as operações de retirar e devolver um livro. Como os parâmetros quantidade máxima de livros e prazo máximo para empréstimo podem variar conforme o tipo de usuário e seu estado, ao invés de tratá-los como atributos da classe vamos definir dois métodos – `getCotaMaxima()` e `getPrazoMaximo()` – que ficarão responsáveis pela determinação desses parâmetros no momento em que sejam necessários. Na classe Usuario esses métodos retornam valores constantes, conforme abaixo:

```
public int getCotaMaxima(){
    return 2;
}

public int getPrazoMaximo(){
    return 4;
}
```

Posteriormente, na definição das classes `UsuarioAluno` e `UsuarioProfessor` esses métodos serão redefinidos de acordo com as especificações para cada tipo de usuário.

De uma maneira geral, o comportamento de um objeto depende de seu estado, o que nos exige saber o estado do objeto nos métodos que implementam suas operações. Muitas vezes a determinação do estado de um objeto não é trivial, o que pode obscurecer a implementação das operações e propiciar inconsistências na determinação do estado por diferentes métodos. Torna-se conveniente, portanto, definirmos métodos auxiliares que sejam responsáveis por determinar, a qualquer momento, o estado do objeto de uma forma simples e segura. Para cada estado previsto no projeto da classe definimos um método de nome `isNomeDoEstado()`, que retorna uma condição indicando se o objeto se encontra no estado `NomeDoEstado`.

No caso da classe usuário iremos definir os métodos `isAptoARetirar()` e `isADevolver()`, de acordo com o diagrama de estados já construído.

```
public boolean isADevolver() {
    return((this.livrosRetirados.size()>= this.getCotaMaxima()
            || this.temPrazoVencido()) ? true:false );
}

public boolean isAptoARetirar() {
    return(!this.isADevolver());
}
```

A expressão `livrosRetirados.size()` será, portanto, igual ao número de livros retirados pelo usuário.

O método `temPrazoVencido()` irá verificar se algum dos livros retirados pelo usuário está com seu prazo de devolução vencido. Para isso será necessário a colaboração da classe `Livro`, responsável por controlar a situação de cada livro individualmente.

O código abaixo verifica se há algum livro em atraso, na lista de livros retirados pelo usuário:

```
public boolean temPrazoVencido () {
    Livro livro;
    Iterator it = livrosRetirados.iterator();
    while(it.hasNext()) {
        livro = it.next();
        if(livro.isEmAtraso()){
            return true;
        }
    }
    return false;
}
```

Iremos definir o método `isEmAtraso()`, na classe `Livro`, de forma similar aos métodos `isAptoARetirar()` e `isADevolver()` da classe `Usuario`.
Com o suporte dos métodos definidos acima, a implementação dos métodos `retiraLivro()` e `devolveLivro()` pode ser feita com facilidade, a partir dos diagramas de atividades correspondentes.

```
public boolean retiraLivro(Livro it) {
    if(this.isAptoARetirar()){
        if(it.empresta(this, getPrazoMaximo())){
            this.livrosRetirados.add(it);
            return true;
        }
        else{
            return false;
        }
    }
    else{
        return false;
    }
}

public boolean devolveLivro (Livro it) {
    if (it.retorna(this)){
        this.livrosRetirados.removeElement(it);
        return true;
    }
    else{
        return false;
    }
}
```

Os métodos `empresta()` e `retorna()` serão definidos na classe `Livro`, para implementar a parte dessas operações de responsabilidade daquela classe, conforme os diagramas acima mencionados.

Para concluir a definição dessa classe iremos acrescentar alguns métodos auxiliares que permitem a objetos de qualquer classe observar partes do estado do objeto:

`isProfessor()` de acordo com o diagrama de atividades para a operação empréstimo de periódico, a classe `Periodico` irá precisar testar se o usuário que solicita uma retirada é um professor. Para simplificar essa tarefa iremos definir o método `isProfessor()` na classe `Usuario` retornando sempre `false` e redefini-lo na classe `UsuarioProfessor` para retornar `true`;

`getNome()` permite que objetos de outras classes possam obter o nome do usuário, mantendo o atributo nome como `private`;

`toString()` fornece uma `String` com a identificação do usuário (categoria de usuário e nome);

`listaCarga()` imprime ficha com os dados do usuário e a lista dos livros de posse do mesmo.

```
public boolean isProfessor(){
    return false;
}

public String getNome(){
    return this.nome;
}

public String toString(){
    return "Usuario "+nome;
}
```

```
public void listaCarga(){
    System.out.println(this.toString()+" Limite: "+ this.getCotaMaxima()+
                       " Carga atual: " + this.livrosRetirados.size());
    Iterator it = this.livrosRetirados.iterator();
    while (it.hasNext()){
        System.out.println(it.next());
    }
}
```

6.2. Classe `UsuarioAluno`

No projeto da classe `UsuarioAluno` foi definido um único atributo especializado, que é a data de expiração do cadastro. Para armazenar essa data utilizaremos a classe `Date`, do pacote `java.util`, que oferece facilidades para manuseio de datas. A única operação especializada é a renovação do cadastro, que simplesmente substitui a data de expiração do cadastro.

Podemos, portanto, iniciar a definição da classe com:

```
import java.util.Date;

public class UsuarioAluno extends Usuario {
    private Date dataLimite;

    public UsuarioAluno(String st, Date dt) {
        super(st);
        this.dataLimite=dt;
    }

    public void renovaCartao (Date dt) {
        this.dataLimite=dt;
    }

}
```

Tal como foi feito para a classe `Usuario`, iremos definir dois métodos auxiliares para a determinação do estado atual do aluno:

```
public boolean isRegular(){
    Date hoje=new Date();
    return dataLimite.after(hoje);
}

public boolean isARenovar(){
    return !isRegular();
}
```

O comando `new Date()` cria um objeto do tipo `Date` com a data e hora atual. O método `after()`, da classe `Date`, retorna `true` se a data armazenada no objeto é posterior à data fornecida como parâmetro.

Podemos agora redefinir os métodos `getCotaMaxima()` e `getPrazoMaximo()`, de acordo com os critérios aplicáveis a um aluno:

```
public int getCotaMaxima(){
    return (isRegular() ? 3 : super.getCotaMaxima());
}
```

```
public int getPrazoMaximo(){
    return (isRegular()? 7 : super.getPrazoMaximo());
}
```

Resta, finalmente, apenas o método `toString()` a ser redefinido:

```
public String toString () {
    return("Aluno " + getNome());
}
```

6.3. Classe `UsuarioProfessor`

No projeto da classe `UsuarioProfessor` são definidas apenas as operações especializadas de bloqueio e desbloqueio de livros. Nos dois casos a única responsabilidade de um objeto da classe `UsuarioProfessor` é chamar os métodos correspondentes do objeto da classe `Livro`.

Podemos, portanto, iniciar a definição da classe com:

```
public class UsuarioProfessor extends Usuario{

    public UsuarioProfessor(String st) {
        super(st);
    }

    public boolean bloqueiaLivro(Livro it, int prazo) {
        return it.bloqueia((Usuario)this, prazo);
    }

    public boolean desbloqueiaLivro (Livro it) {
        return it.desbloqueia((Usuario)this);
    }
}
```

Note que o objeto `this`, nessa classe, é do tipo `UsuarioProfessor` e não `Usuario`. Para uniformizar a interface da classe `Livro`, estamos promovendo o tipo do objeto para Usuario nas chamadas dos métodos `bloqueia()` e `desbloqueia()`.

Para concluir a definição dessa classe resta redefinir os métodos abaixo:

```
public int getCargaLimite(){
    return 5;
}

public int getPrazoMaximo(){
    return 14;
}

public boolean isProfessor(){
    return true;
}

public String toString(){
    return "Prof. "+super.getNome();
}
```

6.4. Classe `Livro`

No projeto da classe `Livro` são definidos como atributos o título do livro, dados do empréstimo e dados do bloqueio. Os atributos do empréstimo e do bloqueio são: usuário que realizou a operação, data da operação e data prevista para devolução ou desbloqueio.

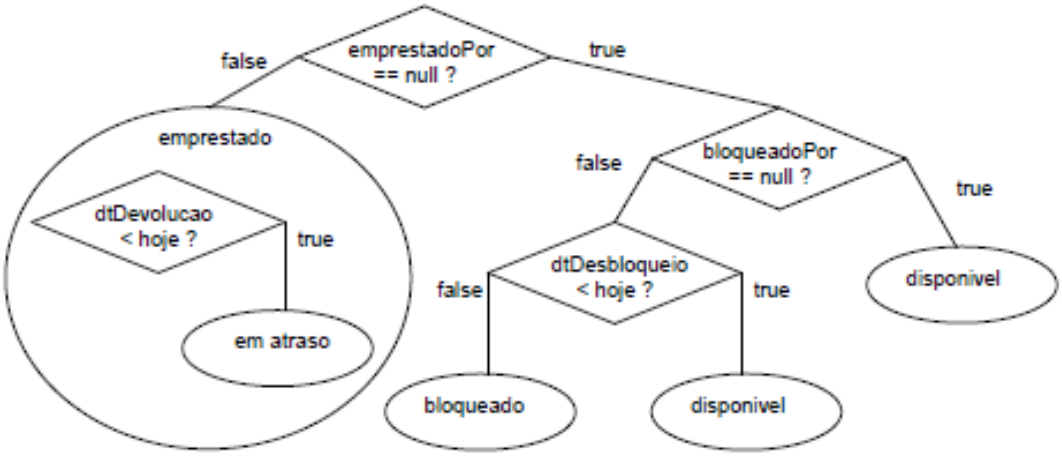
Tal como foi feito para a classe `Usuario`, iremos definir métodos auxiliares para determinar o estado de um `Livro`.

Podemos, portanto, iniciar a definição da classe com a seguinte estrutura geral:

```
import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.Date;

public class Livro {
    private String titulo;
    private Usuario retiradoPor;
    private Date dtEmprestimo;
    private Date dtDevolucao;
    private Usuario bloqueadoPor;
    private Date dtBloqueio;
    private Date dtDesbloqueio;

    public Livro (String tit) {
      this.titulo = tit;
    }

    public boolean isDisponivel() {

    }

    public boolean isEmprestado() {

    }

    public boolean isBloqueado() {

    }

    public boolean isEmAtraso() {

    }

    public boolean bloqueia (Usuario u,int prazo){

    }

    public boolean desbloqueia (Usuario u) {

    }

    public boolean empresta (Usuario u, int prazo) {

    }

    public boolean retorna (Usuario u) {

    }

}
```

Para determinar o estado atual de um livro, iremos utilizar a árvore de decisões da figura abaixo. Esse tipo de árvore nos permite definir o estado de forma perfeitamente

determinista, sem ambiguidade ou indefinições. Note que a condição "em atraso" é subordinada à condição "emprestado" – um livro só pode estar em atraso se está emprestado. Dizemos que o estado "em atraso" é um *sub-estado* de "emprestado". Sendo assim, podemos deixar a árvore que contém "em atraso" incompleta, já que não estamos interessados no estado "emprestado E em dia".



O código seguinte implementa os métodos auxiliares com base nessa árvore de decisões.

```
public boolean isDisponivel() {
    Date hoje=new Date();
    return this.retiradoPor==null&&
           (this.bloqueadoPor==null||this.dtDesbloqueio.before(hoje));
}

public boolean isEmprestado() {
    return !(this.retiradoPor==null);
}

public boolean isBloqueado() {
    Date hoje=new Date();
    return this.retiradoPor==null&&
           !(this.bloqueadoPor==null)&&
           !(this.dtDesbloqueio.before(hoje));
}
```

Com esses métodos já definidos, podemos implementar as operações principais com facilidade:

```
public boolean bloqueia (Usuario u,int prazo){
    GregorianCalendar cal = new GregorianCalendar();
    if(this.isDisponivel()&& u.isProfessor()) {
        this.bloqueadoPor=u;
        this.dtBloqueio=cal.getTime();
        cal.add(Calendar.DATE, (prazo>20?20:prazo));
        this.dtDesbloqueio=cal.getTime();
        return true;
    }
    return false;
}

public boolean desbloqueia (Usuario u) {
    if (u == this.bloqueadoPor) {
        this.bloqueadoPor=null;
        return true;
    }
      return false;
}
```

```
public boolean empresta(Usuario u, int prazo) {
    GregorianCalendar cal = new GregorianCalendar();
    if (this.isDisponivel()) {
        this.retiradoPor = u;
        this.dtEmprestimo = cal.getTime();
        cal.add(Calendar.DATE, prazo);
        this.dtDevolucao=cal.getTime();
        return true;
    }
    return false;
}

public boolean retorna (Usuario u) {
    if (u == this.retiradoPor) {
        this.retiradoPor=null;
        return true;
    }
    return false;
}
```

Para completar a definição da classe iremos definir o método `toString()` para fornecer um `String` com o título do livro e sua situação atual:

```
public String toString() {
    String st = new String();
    if(isDisponivel()){
        return this.titulo+" disponivel";
    }
    if(isEmprestado()){
        st = " retirado por " + retiradoPor + " em " + dma(dtEmprestimo)+
             " ate " + dma(dtDevolucao);
    }
    else{
        st = " bloqueado por " + bloqueadoPor + " em " + dma(dtBloqueio)+
             " ate " + dma(dtDesbloqueio);
    }
    return titulo+st;
}

private String dma(Date dt) {
    GregorianCalendar cal = new GregorianCalendar();
    cal.setTime(dt);
    return  cal.get(Calendar.DATE) +"/"+
            (cal.get(Calendar.MONTH)+1)+"/"+
            cal.get(Calendar.YEAR);
}
```

O método auxiliar `dma()` transforma uma data numa `String` no formato dia / mês /ano. Como é utilizado apenas pela classe `Livro`, foi definido como `private`.

6.5. Classe `Periodico`

No projeto da classe `Periodico` é definida apenas uma operação de empréstimo especializada.

O código seguinte implementa essa classe:

```
public class Periodico extends Livro {

    public Periodico (String tit) {
        super(tit);
    }

    public boolean empresta (Usuario u, int prazo) {
        if (u.isProfessor()){
            return super.empresta (u, 7);
        }
        else{
            return false;
        }
    }
}
```

O método `empresta()` apenas verifica se o usuário é um professor, utilizando para isso o método `isProfessor()`, definido nas classes `Usuario` e `UsuarioProfessor`, e, em caso afirmativo, executa o método de empréstimo da superclasse, com o prazo de empréstimo fixado em 7 dias.

Atividades Propostas

Em dupla, fazer a análise do código apresentado. E, em seguida, executar as seguintes tarefas.

Tarefa 01 Identifique no pacote de classes, os casos de reescrita de métodos.

Tarefa 02 Identifique um caso de polimorfismo no conjunto de códigos apresentados.

Tarefa 03 Explique a importância de termos métodos auxiliares na codificação de classes.

Tarefa 04 O que você entende por mudança de estado de um objeto? Em que situação um objeto muda de estado? Por que a construção de diagramas de estados facilita a codificação?

Tarefa 05 Qual a importância da elaboração de diagramas de atividades antes da codificação dos métodos?

Tarefa 06 Vamos supor que a biblioteca passe a manter DVDs em seu acervo, para os quais são previstas operações de empréstimo de acordo com as seguintes regras específicas:

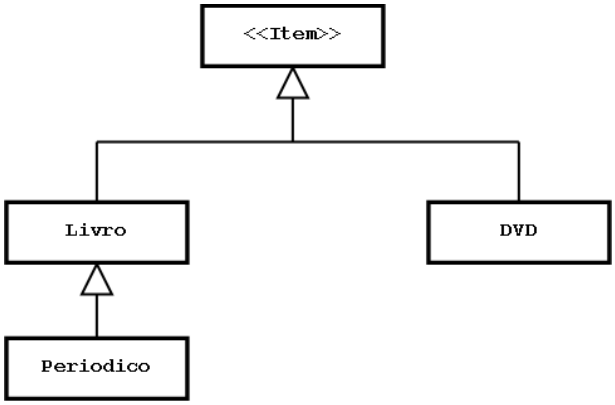
- a cada DVD é associado um nível de privilégio, que determina que tipos de usuários podem retirá-lo, podendo ser: apenas professores; alunos ou professores ou qualquer usuário;
- o nível de privilégio de um DVD pode ser alterado a qualquer momento;
- o prazo de empréstimo de um DVD é sempre de dois dias;
- os DVDs não podem ser bloqueados/desbloqueados.

Para isso, é preciso definir um tipo abstrato de dados para os DVD, através das operações: empréstimo, retorno e alteração de nível de privilégio. Para implementar esse tipo abstrato de dados você irá definir a classe **DVD**.

A questão a ser decidida é: devemos incluir essa nova classe na hierarquia de classes existente? Como isso deve ser feito? É possível estabelecer uma verdadeira relação de generalização/especialização entre DVD e Livro?

Sua resposta deve ter sido: "não é possível estabelecer uma relação de herança entre DVD e Livro." Entretanto, As operações de empréstimo e retorno são comuns aos dois tipos, podendo ser vantajoso para a aplicação tratá-los como um único tipo, ao menos sob esses aspectos. Então, **como implementar essa situação?**

As **classes abstratas** nos fornecem uma solução adequada para situações como essa. Podemos criar uma classe abstrata `Item`, englobando todos os itens do acervo da biblioteca, como superclasse de `Livro` e `DVD`.



A única função da classe `Item` é servir como superclasse, não podendo existir na aplicação objetos instanciados diretamente dela. Sendo assim, podemos omitir de sua implementação tudo o que for específico de suas subclasses, e nos restringir apenas às estruturas de dados e métodos que serão de fato herdados tanto pela classe `Livro` como `DVD`.

Considere os detalhes da operação **empréstimo** para os tipos `Livro` e `DVD`:

- Para que um livro possa ser emprestado, basta que ele esteja disponível. Para um DVD, é necessário também que o usuário que o solicita seja de um tipo compatível com o seu nível de privilégio;
- Para que um DVD seja considerado disponível, basta que ela não esteja emprestado. Para um livro, é necessário também que não esteja bloqueado.

Veja que não se pode incluir na definição da classe `Item` nenhuma dessas duas implementações possíveis para a operação de empréstimo. Em situações como essa é preciso definir uma operação empréstimo mais genérica, que possa ser estendida pelas classes `Livro` e `DVD`, e implementar essa operação na classe abstrata `Item`.

Tomando por base a exposição acima, implemente a classe abstrata `Item`. Faça as modificações necessárias no pacote de classes fornecido. Identifique os atributos e métodos que devem estar na superclasse abstrata. Identifique também se há necessidade de definir algum método abstrato.

Tarefa 07 Elabore um programa principal de controle de bibliotecas, fazendo uso do pacote de classes implementadas. Seu programa deve trabalhar em dois modos: administrador e atendimento. No modo administrador, devem ser oferecidas opções para cadastrar e remover livros, periódicos, DVDs e usuários. No modo de atendimento, o sistema oferece as funcionalidades de empréstimo, devolução, bloqueio e desbloqueio de livros. O atendimento deve ser iniciado com a solicitação de um número de usuário e, em seguida, devem ser apresentadas as opções para aquele tipo de usuário: retirada de livros, devolução e consulta da situação do usuário. Adicionalmente, devem ser oferecidas as opções de bloqueio e desbloqueio para professores. Nas operações de retirada, devolução, bloqueio e desbloqueio o programa exibe a situação do livro (ou DVD, quando for o caso) antes da operação e, caso a mesma seja aceita, o seu estado após a operação.