```
public class Aluno extends Usuario {
  private String matricula;

  public Aluno(String nome, int cpf, String genero, String matricula) {
    super(nome, cpf, genero, "Aluno");
    this.matricula = matricula;
  }

  public String getMatricula() {
    return matricula;
  }

  public void setMatricula(String matricula) {
    this.matricula = matricula;
  }

  @Override
  public String toString() {
    return "Aluno{" +
        "nome='" + nome + '\'' +
        ", cpf=" + cpf +
        ", genero='" + genero + '\'' +
        ", tipoUsuario='" + tipoUsuario + '\'' +
        ", matricula='" + matricula + '\'' +
        '}';
  }
}
public class Professor extends Usuario {
  private String especialidade;

  public Professor(String nome, int cpf, String genero, String especialidade) {
    super(nome, cpf, genero, "Professor");
    this.especialidade = especialidade;
  }

  public String getEspecialidade() {
    return especialidade;
  }

  public void setEspecialidade(String especialidade) {
    this.especialidade= especialidade;
  }
  @Override
  public String toString() {
    return "Professor{" +
        "nome='" + nome + '\'' +
        ", cpf=" + cpf +
        ", genero='" + genero + '\'' +
```

```
                ", tipoUsuario='" + tipoUsuario + '\'' +
                ", Especialidade='" + especialidade + '\'' +
                '}';
    }

    // Métodos específicos de Professor, se necessário
}
public abstract class Pessoa {
    protected String nome;
    protected int cpf;
    protected String genero;

    public Pessoa(String nome, int cpf, String genero) {
        this.nome = nome;
        this.cpf = cpf;
        this.genero = genero;
    }
    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public int getCpf() {
        return cpf;
    }

    public void setCpf(int cpf) {
        this.cpf = cpf;
    }

    public String getGenero() {
        return genero;
    }

    public void setGenero(String genero) {
        this.genero = genero;
    }

    @Override
    public String toString() {
        return "Pessoa{" +
                "nome='" + nome + '\'' +
                ", cpf=" + cpf +
                ", genero='" + genero + '\'' +
                '}';
```

```
    }
}
public class Livro {
    private String titulo;
    private String autor;
    private int anoPublicacao;
    private boolean estaEmprestado;

    public Livro(String titulo, String autor, int anoPublicacao) {
        this.titulo = titulo;
        this.autor = autor;
        this.anoPublicacao = anoPublicacao;
        this.estaEmprestado = false;
    }

    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public String getAutor() {
        return autor;
    }

    public void setAutor(String autor) {
        this.autor = autor;
    }

    public int getAnoPublicacao() {
        return anoPublicacao;
    }

    public void setAnoPublicacao(int anoPublicacao) {
        this.anoPublicacao = anoPublicacao;
    }

    public boolean isEstaEmprestado() {
        return estaEmprestado;
    }

    public void setEstaEmprestado(boolean estaEmprestado) {
        this.estaEmprestado = estaEmprestado;
    }

    @Override
```

```
    public String toString() {
        return "Livro{" +
                "titulo='" + titulo + '\'' +
                ", autor='" + autor + '\'' +
                ", anoPublicacao=" + anoPublicacao +
                ", estaEmprestado=" + estaEmprestado +
                '}';
    }
}
public class Usuario extends Pessoa {
    String tipoUsuario;

    public Usuario(String nome, int cpf, String genero, String tipoUsuario) {
        super(nome, cpf, genero);
        this.tipoUsuario = tipoUsuario;
    }

    public String getTipoUsuario() {
        return tipoUsuario;
    }

    public void setTipoUsuario(String tipoUsuario) {
        this.tipoUsuario = tipoUsuario;
    }


    @Override
    public String toString() {
        return super.toString() +
                ", tipoUsuario='" + tipoUsuario + '\'' +
                '}';
    }


}
public interface Emprestavel {
    void emprestarLivro();

    void devolverLivro();
}
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;

public class Emprestimo {
    private Livro livroEmprestado;
    private Usuario usuarioEmprestimo;
    private LocalDateTime dataEmprestimo;
```

```
    private LocalDateTime dataPrevistaDevolucao;
    private LocalDateTime dataDevolucao;

    public Emprestimo(Livro livroEmprestado, Usuario usuarioEmprestimo, LocalDateTime
dataEmprestimo, LocalDateTime dataPrevistaDevolucao) {
        this.livroEmprestado = livroEmprestado;
        this.usuarioEmprestimo = usuarioEmprestimo;
        this.dataEmprestimo = dataEmprestimo;
        this.dataPrevistaDevolucao = dataPrevistaDevolucao;
    }

    public Livro getLivroEmprestado() {
        return livroEmprestado;
    }

    public Usuario getUsuarioEmprestimo() {
        return usuarioEmprestimo;
    }

    public LocalDateTime getDataEmprestimo() {
        return dataEmprestimo;
    }

    public LocalDateTime getDataPrevistaDevolucao() {
        return dataPrevistaDevolucao;
    }

    public LocalDateTime getDataDevolucao() {
        return dataDevolucao;
    }

    public void setDataDevolucao(LocalDateTime dataDevolucao) {
        this.dataDevolucao = dataDevolucao;
    }

    public long calcularDiasAtraso() {
        if (dataDevolucao == null) {
            return 0;
        }
        return ChronoUnit.DAYS.between(dataPrevistaDevolucao, dataDevolucao);
    }

    public double calcularMultaPorAtraso() {
        long diasAtraso = calcularDiasAtraso();
        if (diasAtraso <= 0) {
            return 0;
        }
        // Exemplo de cálculo de multa: R$ 1,00 por dia de atraso
```

```
        return diasAtraso * 1.0;
    }

    @Override
    public String toString() {
        return "Emprestimo{" +
                "livroEmprestado=" + livroEmprestado.getTitulo() +
                ", usuarioEmprestimo=" + usuarioEmprestimo.getNome() +
                ", dataEmprestimo=" + formatarData(dataEmprestimo) +
                ", dataPrevistaDevolucao=" + formatarData(dataPrevistaDevolucao) +
                ", dataDevolucao=" + (dataDevolucao != null ? formatarData(dataDevolucao) : "Em
aberto") +
                '}';
    }

    public String formatarData(LocalDateTime data) {
        if (data == null) {
            return "null";
        }
        return data.format(DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm"));
    }
}
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.time.DateTimeException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;
import java.util.*;
import java.time.LocalDate;
import java.util.InputMismatchException;
import java.util.Scanner;
import java.util.List;

public class Biblioteca {
    public int cpfUsuario;
    private static final DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm");

    private static List<Usuario> usuarios;
    private List<Livro> livros;
    private List<Emprestimo> emprestimos;

    public Biblioteca() {
        this.usuarios = new ArrayList<>();
        this.livros = new ArrayList<>();
        this.emprestimos = new ArrayList<>();
```

```
}

public void executar() {
    Scanner scanner = new Scanner(System.in);

    while (true) {
        exibirMenu();
        int opcao = scanner.nextInt();
        scanner.nextLine(); // Limpar o buffer do scanner

        switch (opcao) {
            case 1:
                cadastrarUsuario(scanner);
                break;
            case 2:
                cadastrarLivro(scanner);
                break;
            case 3:
                listarUsuarios();
                break;
            case 4:
                listarLivros();
                break;
            case 5:
                emprestarLivro(scanner);
                break;
            case 6:
                devolverLivro(scanner);
                break;
            case 7:
                deletarLivro(scanner);
                break;
            case 8:
                alterarLivro(scanner);
                break;
            case 9:
                deletarUsuario(scanner);
                break;
            case 10:
                alterarUsuario(scanner);
                break;
            case 11:
                salvarLivrosEmArquivo();
                salvarEmprestimosEmArquivo();
                System.out.println("Saindo do sistema...");
                return;
            default:
                System.out.println("Opção inválida. Escolha novamente.");
```

```
        }
    }
}


private void exibirMenu() {
    System.out.println("\nMenu");
    System.out.println("1. Cadastrar Usuário");
    System.out.println("2. Cadastrar Livro");
    System.out.println("3. Listar Usuários");
    System.out.println("4. Listar Livros");
    System.out.println("5. Emprestar Livro");
    System.out.println("6. Devolver Livro");
    System.out.println("7. Deletar Livro");
    System.out.println("8. Alterar Livro");
    System.out.println("9. Deletar Usuário");
    System.out.println("10. Alterar Usuário");
    System.out.println("11. Sair");
    System.out.print("Escolha uma opção: ");
}

private void cadastrarUsuario(Scanner scanner) {
    System.out.println("\nCadastro de Usuário");
    System.out.print("Nome: ");
    String nome = scanner.nextLine();
    System.out.print("CPF: ");
    String cpfStr = scanner.nextLine();

    // Verifica se o CPF possui apenas dígitos
    if (!cpfStr.matches("\\d+")) {
        System.out.println("CPF inválido. Digite apenas números.");
        return;
    }

    int cpf = Integer.parseInt(cpfStr);

    System.out.print("Gênero: ");
    String genero = scanner.nextLine();
    System.out.print("Tipo de usuário (Aluno/Professor): ");
    String tipoUsuario = scanner.nextLine();

    if ("Aluno".equalsIgnoreCase(tipoUsuario)) {
        System.out.print("Matrícula: ");
        String matricula = scanner.nextLine();
        Aluno aluno = new Aluno(nome, cpf, genero, matricula);
        usuarios.add(aluno);
    } else if ("Professor".equalsIgnoreCase(tipoUsuario)) {
        System.out.print("Especialidade: ");
```

```
            String especialidade = scanner.nextLine();
            Professor professor = new Professor(nome, cpf, genero, especialidade);
            usuarios.add(professor);
        } else {
            System.out.println("Tipo de usuário inválido.");
        }
    }

    private void cadastrarLivro(Scanner scanner) {
        System.out.println("\nCadastro de Livro");
        System.out.print("Título: ");
        String titulo = scanner.nextLine();
        System.out.print("Autor: ");
        String autor = scanner.nextLine();
        System.out.print("Ano de Publicação: ");
        int anoPublicacao = scanner.nextInt();
        scanner.nextLine(); // Limpar o buffer do scanner

        Livro livro = new Livro(titulo, autor, anoPublicacao);
        livros.add(livro);
    }

    private void listarUsuarios() {
        System.out.println("\nUsuários Cadastrados");
        if (usuarios.isEmpty()) {
            System.out.println("Não há usuários cadastrados.");
        } else {
            for (Usuario usuario : usuarios) {
                System.out.println(usuario.toString());
                System.out.println("-----------------------------------");
            }
        }
    }


    private void listarLivros() {
        System.out.println("\nLivros Cadastrados");
        if (livros.isEmpty()) {
            System.out.println("Não há livros cadastrados.");
        } else {
            for (Livro livro : livros) {
                String status = livro.isEstaEmprestado() ? "sim" : "não";
                System.out.print("titulo: "+livro.getTitulo() + "\t- autor : " + livro.getAutor() + "\t- ano
publicado:" + livro.getAnoPublicacao()+"\t- Esta emprestado :"+ status);
                if (livro.isEstaEmprestado()) {
                    System.out.println(" - Emprestado para: " +
encontrarUsuarioQueEmprestou(livro));
                } else {
```

```
                System.out.println();
            }
        }
    }
}

private String encontrarUsuarioQueEmprestou(Livro livro) {
    for (Emprestimo emprestimo : emprestimos) {
        if (emprestimo.getLivroEmprestado().equals(livro)) {
            return emprestimo.getUsuarioEmprestimo().getNome();
        }
    }
    return "Desconhecido";
}

private void emprestarLivro(Scanner scanner) {
    // Verificar se há livros cadastrados para emprestar
    if (livros.isEmpty()) {
        System.out.println("Não há livros cadastrados para emprestar.");
        return;
    }

    // Listar usuários disponíveis para o empréstimo
    listarUsuarios();
    System.out.print("Escolha o usuário pelo CPF para realizar o empréstimo: ");

    int cpfUsuario;
    try {
        cpfUsuario = scanner.nextInt();
    } catch (InputMismatchException e) {
        System.out.println("Formato inválido para CPF. Digite apenas números.");
        scanner.nextLine(); // Limpar o buffer do scanner
        return;
    }
    scanner.nextLine(); // Limpar o buffer do scanner

    // Verificar se o usuário com o CPF fornecido existe
    Usuario usuarioEmprestimo = null;
    for (Usuario usuario : usuarios) {
        if (usuario.getCpf() == cpfUsuario) {
            usuarioEmprestimo = usuario;
            break;
        }
    }

    // Verificar se o usuário foi encontrado
    if (usuarioEmprestimo == null) {
        System.out.println("Usuário não encontrado com CPF " + cpfUsuario);
```

```
            return;
        }

        // Listar os livros disponíveis para empréstimo
        listarLivros();
        System.out.print("Escolha o livro pelo título para emprestar: ");
        String tituloLivro = scanner.nextLine();

        // Encontrar o livro com o título fornecido e que não esteja emprestado
        Livro livroEmprestimo = null;
        for (Livro livro : livros) {
            if (livro.getTitulo().equalsIgnoreCase(tituloLivro) && !livro.isEstaEmprestado()) {
                livroEmprestimo = livro;
                livro.setEstaEmprestado(true); // Marcar o livro como emprestado
                break;
            }
        }

        // Verificar se o livro foi encontrado e está disponível para empréstimo
        if (livroEmprestimo == null) {
            System.out.println("Livro não encontrado ou já emprestado.");
            return;
        }

        // Obter e validar a data de empréstimo
        LocalDateTime dataEmprestimo;
        try {
            System.out.println("Digite a data de empréstimo (dd/MM/yyyy): ");
            System.out.print("Dia: ");
            int dia = scanner.nextInt();
            System.out.print("Mês: ");
            int mes = scanner.nextInt();
            System.out.print("Ano: ");
            int ano = scanner.nextInt();
            scanner.nextLine(); // Limpar o buffer do scanner

            // Criar o objeto LocalDateTime para a data de empréstimo
            dataEmprestimo = LocalDateTime.of(ano, mes, dia, 0, 0);

            // Verificar se a data de empréstimo é válida (opcional, dependendo da lógica do
sistema)
            LocalDateTime dataAtual = LocalDateTime.now();
            if (dataEmprestimo.isBefore(dataAtual)) {
                System.out.println("Data de empréstimo não pode ser no passado.");
                livroEmprestimo.setEstaEmprestado(false); // Desfazer marcação de emprestado
                return;
            }
```

```
        } catch (DateTimeException | InputMismatchException e) {
            System.out.println("Formato inválido para data. Digite apenas números.");
            return;
        }

        // Calcular a data prevista de devolução (exemplo: devolução em 7 dias)
        LocalDateTime dataPrevistaDevolucao = dataEmprestimo.plusDays(7);

        // Criar o objeto de empréstimo e adicionar à lista de empréstimos
        Emprestimo emprestimo = new Emprestimo(livroEmprestimo, usuarioEmprestimo,
dataEmprestimo, dataPrevistaDevolucao);
        emprestimos.add(emprestimo);

        // Mensagem de confirmação
        System.out.println("Livro emprestado com sucesso para " +
usuarioEmprestimo.getNome() + ".");
    }

    private void devolverLivro(Scanner scanner) {
        if (emprestimos.isEmpty()) {
            System.out.println("Não há livros emprestados para devolver.");
            return;
        }

        listarEmprestimos();

        System.out.print("Digite o índice do empréstimo que deseja devolver: ");
        int indiceEmprestimo = scanner.nextInt();
        scanner.nextLine(); // Limpar o buffer do scanner

        if (indiceEmprestimo < 0 || indiceEmprestimo >= emprestimos.size()) {
            System.out.println("Índice inválido.");
            return;
        }

        Emprestimo emprestimo = emprestimos.get(indiceEmprestimo);

        System.out.println("Digite a data de devolução do livro (dd/MM/yyyy): ");
        System.out.print("Dia: ");
        int dia;
        int mes;
        int ano;
        try {
            dia = scanner.nextInt();
            System.out.print("Mês: ");
            mes = scanner.nextInt();
            System.out.print("Ano: ");
            ano = scanner.nextInt();
```

```
        } catch (InputMismatchException e) {
            System.out.println("Formato inválido para data. Digite apenas números.");
            scanner.nextLine(); // Limpar o buffer do scanner
            return;
        }
        scanner.nextLine(); // Limpar o buffer do scanner

        LocalDate dataDevolucao = LocalDate.of(ano, mes, dia);

        emprestimo.setDataDevolucao(dataDevolucao.atStartOfDay());
        emprestimo.getLivroEmprestado().setEstaEmprestado(false);

        LocalDate dataPrevistaDevolucao =
LocalDate.from(emprestimo.getDataPrevistaDevolucao());
        long diasAtraso = ChronoUnit.DAYS.between(dataPrevistaDevolucao, dataDevolucao);

        if (diasAtraso > 0) {
            double multa = diasAtraso * 10.0; // R$ 10,00 por dia de atraso
            System.out.println("Livro devolvido com atraso de " + diasAtraso + " dias.");
            System.out.println("Multa a ser paga: R$ " + multa);

            // Lógica adicional para aplicar multas ao usuário, se desejado
        } else {
            System.out.println("Livro devolvido dentro do prazo.");
        }

        emprestimos.remove(indiceEmprestimo);
    }




    private void listarEmprestimos() {
        System.out.println("\nEmpréstimos Ativos");
        if (emprestimos.isEmpty()) {
            System.out.println("Não há livros emprestados no momento.");
        } else {
            for (int i = 0; i < emprestimos.size(); i++) {
                Emprestimo emprestimo = emprestimos.get(i);
                System.out.println("Índice: " + i);
                System.out.println("Livro: " + emprestimo.getLivroEmprestado().getTitulo());
                System.out.println("Usuário: " + emprestimo.getUsuarioEmprestimo().getNome());
                System.out.println("Data do Empréstimo: " +
emprestimo.formatarData(emprestimo.getDataEmprestimo()));
                System.out.println("Data Prevista de Devolução: " +
emprestimo.formatarData(emprestimo.getDataPrevistaDevolucao()));
                System.out.println("----------------------------------------");
            }
```

```
    }
}
private void deletarLivro(Scanner scanner) {
    System.out.println("\nDeletar Livro");
    listarLivros();

    if (livros.isEmpty()) {
        System.out.println("Não há livros cadastrados para deletar.");
        return;
    }

    System.out.print("Digite o título do livro que deseja deletar: ");
    String tituloLivro = scanner.nextLine();

    Iterator<Livro> iterator = livros.iterator();
    while (iterator.hasNext()) {
        Livro livro = iterator.next();
        if (livro.getTitulo().equalsIgnoreCase(tituloLivro)) {
            iterator.remove();
            System.out.println("Livro deletado com sucesso.");
            return;
        }
    }

    System.out.println("Livro não encontrado.");
}

private void alterarLivro(Scanner scanner) {
    System.out.println("\nAlterar Livro");
    listarLivros();

    if (livros.isEmpty()) {
        System.out.println("Não há livros cadastrados para alterar.");
        return;
    }

    System.out.print("Digite o título do livro que deseja alterar: ");
    String tituloLivro = scanner.nextLine();

    for (Livro livro : livros) {
        if (livro.getTitulo().equalsIgnoreCase(tituloLivro)) {
            System.out.print("Novo Título: ");
            String novoTitulo = scanner.nextLine();
            System.out.print("Novo Autor: ");
            String novoAutor = scanner.nextLine();
            System.out.print("Novo Ano de Publicação: ");
            int novoAnoPublicacao = scanner.nextInt();
            scanner.nextLine(); // Limpar o buffer do scanner
```

```
            livro.setTitulo(novoTitulo);
            livro.setAutor(novoAutor);
            livro.setAnoPublicacao(novoAnoPublicacao);
            System.out.println("Livro alterado com sucesso.");
            return;
        }
    }

    System.out.println("Livro não encontrado.");
}

private void deletarUsuario(Scanner scanner) {
    if (usuarios.isEmpty()) {
        System.out.println("Não há usuários cadastrados para deletar.");
        return;
    }

    listarUsuarios();

    System.out.print("Escolha o usuário pelo CPF para deletar: ");
    int cpfUsuario;
    try {
        cpfUsuario = scanner.nextInt();
        scanner.nextLine(); // Limpar o buffer do scanner
    } catch (InputMismatchException e) {
        System.out.println("Formato inválido para CPF. Digite apenas números.");
        scanner.nextLine(); // Limpar o buffer do scanner
        return;
    }

    // Encontrar o usuário com o CPF fornecido
    Usuario usuarioParaDeletar = null;
    for (Usuario usuario : usuarios) {
        if (usuario.getCpf() == cpfUsuario) {
            usuarioParaDeletar = usuario;
            break;
        }
    }

    // Verificar se o usuário foi encontrado
    if (usuarioParaDeletar == null) {
        System.out.println("Usuário não encontrado com o CPF fornecido.");
        return;
    }

    // Verificar se o usuário possui livro(s) emprestado(s)
    boolean temLivroEmprestado = false;
```

```
        for (Emprestimo emprestimo : emprestimos) {
            if (emprestimo.getUsuarioEmprestimo().equals(usuarioParaDeletar)) {
                temLivroEmprestado = true;
                break;
            }
        }

        if (temLivroEmprestado) {
            System.out.println("O usuário está com livro(s) emprestado(s). Favor devolver o(s)
livro(s) antes de deletar o usuário.");
        } else {
            // Remover o usuário da lista
            usuarios.remove(usuarioParaDeletar);
            System.out.println("Usuário deletado com sucesso.");
        }
    }


    private void alterarUsuario(Scanner scanner) {
        // Verificar se há usuários cadastrados para alterar
        if (usuarios.isEmpty()) {
            System.out.println("Não há usuários cadastrados para alterar.");
            return;
        }
        listarUsuarios();

        System.out.println("\nAlteração de Usuário");
        System.out.print("Digite o CPF do usuário que deseja alterar: ");
        int cpfUsuario;
        try {
            cpfUsuario = scanner.nextInt();
            scanner.nextLine(); // Limpar o buffer do scanner
        } catch (InputMismatchException e) {
            System.out.println("Formato inválido para CPF. Digite apenas números.");
            scanner.nextLine(); // Limpar o buffer do scanner
            return;
        }

        // Encontrar o usuário com o CPF especificado
        Usuario usuarioAlterar = null;
        for (Usuario usuario : usuarios) {
            if (usuario.getCpf() == cpfUsuario) {
                usuarioAlterar = usuario;
                break;
            }
        }

        // Verificar se o usuário foi encontrado
```

```
        if (usuarioAlterar == null) {
            System.out.println("Usuário não encontrado para alterar.");
            return;
        }

        // Exibir informações atuais do usuário
        System.out.println("Informações atuais do usuário:");
        System.out.println(usuarioAlterar);

        // Capturar novas informações do usuário
        System.out.print("Novo Nome (atual: " + usuarioAlterar.getNome() + "): ");
        String novoNome = scanner.nextLine();
        System.out.print("Novo Gênero (atual: " + usuarioAlterar.getGenero() + "): ");
        String novoGenero = scanner.nextLine();

        // Atualizar as informações do usuário
        usuarioAlterar.setNome(novoNome);
        usuarioAlterar.setGenero(novoGenero);

        // Verificar e atualizar informações específicas para Aluno ou Professor
        if (usuarioAlterar instanceof Aluno) {
            Aluno alunoAlterar = (Aluno) usuarioAlterar;
            System.out.print("Nova Matrícula (atual: " + alunoAlterar.getMatricula() + "): ");
            String novaMatricula = scanner.nextLine();
            alunoAlterar.setMatricula(novaMatricula);
        } else if (usuarioAlterar instanceof Professor) {
            Professor professorAlterar = (Professor) usuarioAlterar;
            try {
                System.out.print("Nova Especialidade (atual: " +
professorAlterar.getEspecialidade() + "): ");
                String novaEspecialidade = scanner.nextLine();
                professorAlterar.setEspecialidade(novaEspecialidade);
            } catch (InputMismatchException e) {
                System.out.println("Formato inválido para especialidade. Digite apenas letras.");
                scanner.nextLine(); // Limpar o buffer do scanner
                return;
            }
        }

        System.out.println("Usuário alterado com sucesso!");
    }
    private void salvarLivrosEmArquivo() {
        String arquivoLivros = "livros.txt";
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(arquivoLivros))) {
            for (Livro livro : livros) {
                writer.write(livro.getTitulo() + ";" + livro.getAutor() + ";" + livro.getAnoPublicacao());
                writer.newLine();
            }
```

```
            System.out.println("Livros salvos em arquivo.");
        } catch (IOException e) {
            System.out.println("Erro ao salvar livros em arquivo: " + e.getMessage());
        }
    }

    private void salvarEmprestimosEmArquivo() {
        String arquivoEmprestimos = "emprestimos.txt";
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(arquivoEmprestimos))) {
            for (Emprestimo emprestimo : emprestimos) {
                writer.write(emprestimo.getLivroEmprestado().getTitulo() + ";" +
                        emprestimo.getUsuarioEmprestimo().getCpf() + ";" +
                        emprestimo.getDataEmprestimo().format(formatter) + ";" +
                        emprestimo.getDataPrevistaDevolucao().format(formatter) + ";" +
                        (emprestimo.getDataDevolucao() != null ?
emprestimo.getDataDevolucao().format(formatter) : ""));
                writer.newLine();
            }
            System.out.println("Empréstimos salvos em arquivo.");
        } catch (IOException e) {
            System.out.println("Erro ao salvar empréstimos em arquivo: " + e.getMessage());
        }
    }


    public static void main(String[] args) {
        Biblioteca biblioteca = new Biblioteca();
        biblioteca.executar();
    }
}
```