Projeto SAB pelo CRC Modeling

Código de Produção da Versão Preliminar

```
package pSABbyCRC_UnitTestingSuite;

import java.util.HashSet;
import java.util.Iterator;
import java.util.TreeSet;

public class Biblioteca {

     public Biblioteca(String nome) {
          _nome = nome;
          _repositorioLivros = new TreeSet<Livro>();
          _usuarios = new HashSet<Usuario>();
     }

     public void adicionaLivroCatalogo(Livro livro)
               throws AdicionarLivroInexistenteException {
          if (livro != null) {
               livro.setNrCatalogo(this.getNrUnico());
               _repositorioLivros.add(livro);
          } else
               throw new AdicionarLivroInexistenteException(
                         "--->Não pode adicionar livro inexistente!");
     }

     public void registraUsuario(String nome)
               throws UsuarioJaRegistradoException, UsuarioComNomeVazioException,
               UsuarioInexistenteException {
          if (nome != null) {
               if (!nome.isEmpty()) {
                    Usuario usuario = new Usuario(nome);
                    if (!_usuarios.contains(usuario)) {
                         _usuarios.add(usuario);
                    } else
                         throw new UsuarioJaRegistradoException("--->Já existe usuário com o nome \""
                                   + nome + "\"! Use outro nome!");
               } else
                    throw new UsuarioComNomeVazioException("--->Não pode registrar usuario com nome vazio!");
          } else
               throw new UsuarioInexistenteException("--->Não pode registrar usuario inexistente!");
     }

     public void emprestaLivro(Livro livro, Usuario usuario)
               throws LivroIndisponivelParaEmprestimoException, LivroOuUsuarioNulosException{
          if ((livro == null) && (usuario == null))
               throw new LivroOuUsuarioNulosException("--->Livro e Usuário inexistentes!");
          if (livro != null) {
               if (usuario != null) {
                    if (livro.getUsuario() == null) {
                         usuario.anexaLivroAoUsuario(livro);
                         livro.anexaUsuarioAoLivro(usuario);
                    } else
                         throw new LivroIndisponivelParaEmprestimoException("--->Livro " + livro
                                   + " indisponivel para empréstimo!");
               } else
                    throw new LivroOuUsuarioNulosException("--->Usuário inexistente!");
          } else
               throw new LivroOuUsuarioNulosException("--->Não pode emprestar livro inexistente!");
     }

     public void devolveLivro(Livro livro)
               throws DevolveLivroDisponivelParaEmprestimoException, DevolveLivroNuloParaEmprestimoException {
          if (livro != null) {
               Usuario usuario = livro.getUsuario();
               if (usuario != null) {
                    usuario.desanexaLivroDoUsuario(livro);
                    livro.desanexaUsuarioDoLivro();
               } else
                    throw new DevolveLivroDisponivelParaEmprestimoException("---> Tentou devolver livro " + livro
                              + " que está disponivel para empréstimo!");
          } else
               throw new DevolveLivroNuloParaEmprestimoException("--->Não pode emprestar livro inexistente!");
     }

     public Livro buscaLivroPorNrCatalogo(int nrUnico) {
          // nrUnico <= zero devolve nulo: não encontrou livro algum!
          Livro livroAchado = null;
          Iterator<Livro> iter = _repositorioLivros.iterator();
          while ((iter.hasNext() == true) && (livroAchado == null)) {
               Livro livro = (Livro) iter.next();
               int oNrUnico = livro.getNrCatalogo();
               if (oNrUnico == nrUnico)
                    livroAchado = livro;
          }
          return livroAchado;
     }

     public Livro buscaLivroPorTituloAutor(String titulo, String autor)
               throws TituloOuAutorVazioException, TituloOuAutorNuloException {
          Livro livroAchado = null;
          if ((titulo != null) && (autor != null)) {
               if (!titulo.isEmpty() && !autor.isEmpty()) {
                    Iterator<Livro> iter = _repositorioLivros.iterator();
                    while ((iter.hasNext() == true) && (livroAchado == null)) {
                         Livro livro = (Livro) iter.next();
                         String oTitulo = livro.getTitulo();
                         String oAutor = livro.getAutor();
                         if ((oTitulo.equals(titulo)) && (oAutor.equals(autor))) {
                              livroAchado = livro;
                         }
                    }
               } else
                    throw new TituloOuAutorVazioException ("--->Nome do titulo e/ou do autor é(são) vazio(s)<<<");
          } else
               throw new TituloOuAutorNuloException("--->Nome do titulo e/ou do autor é(são) nulo(s)<<<");
          return livroAchado;
     }
```

```
	public Usuario buscaUsuarioPorNome(String nome)
			throws BuscaUsuarioComNomeVazioException, BuscaUsuarioComNomeNuloException {
		Usuario usuarioAchado = null;
		if ((nome != null)) {
			if (!nome.isEmpty()) {
				Iterator<Usuario> iter = _usuarios.iterator();
				while ((iter.hasNext() == true) && (usuarioAchado == null)) {
					Usuario usuario = (Usuario) iter.next();
					String oNome = usuario.getNome();
					if (oNome == nome) {
						usuarioAchado = usuario;
					}
				}
			} else
				throw new BuscaUsuarioComNomeVazioException("--->Nome do usuário é vazio<<<");
		} else
			throw new BuscaUsuarioComNomeNuloException("--->Nome do usuário é nulo<<<");
		return usuarioAchado;
	}

	public void exibeLivrosDisponiveis() {
		System.out.println("Biblioteca: " + _nome);
		System.out.println(">>>Livros Disponiveis para Empréstimo<<<");
		if (_repositorioLivros.size() != 0) {
			Iterator<Livro> iter = _repositorioLivros.iterator();
			while (iter.hasNext() == true) {
				Livro livro = (Livro) iter.next();
				if (livro.getUsuario() == null) {
					livro.exibe();
				}
			}
		} else
			System.out.println("---> Nenhum livro no repositório");
		System.out.println("<<< Livros Disponiveis >>>");
		System.out.println();
	}

	public void exibeLivrosEmprestados() {
		System.out.println("Biblioteca: " + _nome);
		System.out.println(">>>Livros Emprestados<<<");
		if (_repositorioLivros.size() != 0) {
			Iterator<Livro> iter = _repositorioLivros.iterator();
			while (iter.hasNext() == true) {
				Livro livro = (Livro) iter.next();
				if (livro.getUsuario() != null) {
					System.out.println("\t\t"
							+ "-------------------------------------------");
					livro.exibe();
				}
			}
		} else
			System.out.println("---> Nenhum livro no repositório");
		System.out.println("<<< Livros Emprestados >>>");
		System.out.println();
	}

	public void exibeUsuarios() {
		System.out.println("Biblioteca: " + _nome);
		System.out.println(">>>Usuários da Biblioteca<<<");
		if (_usuarios.size() != 0) {
			Iterator<Usuario> iter = _usuarios.iterator();
			while (iter.hasNext() == true) {
				Usuario usuario = (Usuario) iter.next();
				usuario.exibe();
			}
		} else
			System.out.println("---> Nenhum usuário na Biblioteca");
		System.out.println("<<< Usuários >>>");
		System.out.println();
	}

	private int getNrUnico() {
		// Assumo que cada livro recebe um nrUnico diferente
		return _nrUnico = _nrUnico + 1;
	}

	public int sizeRepositorioLivros() {
		return _repositorioLivros.size();
	}

	public int sizeUsuarios() {
		return _usuarios.size();
	}

	private String _nome;
	private int _nrUnico = 0; // _nrUnico > zero!
	private TreeSet<Livro> _repositorioLivros;
	private HashSet<Usuario> _usuarios;
}

package pSABbyCRC_UnitTestingSuite;

public class Livro implements Comparable<Object> {
	public Livro(String titulo, String autor) {
		setTitulo(titulo);
		setAutor(autor);
		// seta _usuario null: livro está disponível para empréstimo:
		desanexaUsuarioDoLivro();
	}

	public void anexaUsuarioAoLivro(Usuario usuario) {
		_usuario = usuario;
	}

	public void desanexaUsuarioDoLivro() {
		anexaUsuarioAoLivro(null);
	}

	public void exibe() {
```

```
            System.out.println("\t\t" + "Titulo: " + "\t\t" + this.getTitulo());
            System.out.println("\t\t" + "Autor: " + "\t\t\t" + this.getAutor());
            System.out.println("\t\t" + "Nr. Catálogo: " + "\t\t"
                        + this.getNrCatalogo());
            if (getUsuario() != null)
                  System.out.println("\t\t" + "Quem Emprestou: " + "\t"
                              + this.getUsuario());
            System.out.println("\t\t"
                        + "~----------------------------------------");
            System.out.println();
      }

      @Override
      public boolean equals(Object obj) {
            return this.compareTo(obj) == 0;
      }

      @Override
      public int compareTo(Object obj) {
            Livro livro = (Livro) obj;
            int livroNrCatalogo = livro.getNrCatalogo();
            int result;
            if (_nrCatalogo < livroNrCatalogo)
                  result = -1;
            else if (_nrCatalogo == livroNrCatalogo)
                  result = 0;
            else
                  result = 1;
            return result;
      }

      @Override
      public int hashCode() {
            Integer integerNrCatalogo = new Integer(_nrCatalogo);
            return integerNrCatalogo.hashCode();
      }

      @Override
      public String toString() {
            return "\"Titulo: " + getTitulo() + " - Autor: " + getAutor() + "\"";
      }

      public int getNrCatalogo() {
            return _nrCatalogo;
      }

      public void setNrCatalogo(int nrCatalogo) {
            _nrCatalogo = nrCatalogo;
      }

      public String getTitulo() {
            return _titulo;
      }

      protected void setTitulo(String titulo) {
            _titulo = titulo;
      }

      public Usuario getUsuario() {
            return _usuario;
      }

      public String getAutor() {
            return _autor;
      }

      protected void setAutor(String autor) {
            _autor = autor;
      }

      private int _nrCatalogo;
      private String _titulo;
      private String _autor;
      // Se _usuario não null: livro está disponível para empréstimo
      // Se _usuario null: livro está emprestado!
      private Usuario _usuario;
}

import java.util.Iterator;
import java.util.LinkedList;

public class Usuario implements Comparable<Object> {
      public Usuario(String nome) {
            setNome(nome);
            _livros = new LinkedList<Livro>();
      }

      public void anexaLivroAoUsuario(Livro livro) {
            _livros.add(livro);
      }

      public void desanexaLivroDoUsuario(Livro livro) {
            _livros.remove(livro);
      }

      public void exibe() {
            System.out.println("\t\t" + "Nome: " + "\t\t" + getNome());
            this.exibeLivrosUsuario();
      }

      private void exibeLivrosUsuario() {
            System.out.println("\t\t" + "\\//Livros emprestados:");
            if (_livros.size() != 0) {
                  Iterator<Livro> iter = _livros.iterator();
                  while (iter.hasNext() == true) {
                        Livro livro = (Livro) iter.next();
                        System.out.println("\t\t\t" + livro.getNrCatalogo() + " "
                                    + livro);
                  }
            } else
```

```
                System.out.println("\t\t" + "---> Nenhum livro emprestado");
            System.out.println("\t\t" + "\\///\\///\\///");
            System.out.println();
        }

        @Override
        public boolean equals(Object obj) {
            return this.compareTo(obj) == 0;

        }

        @Override
        public int compareTo(Object obj) {
            Usuario usuario = (Usuario) obj;
            String nome = usuario.getNome();
            return _nome.compareTo(nome);
        }

        @Override
        public int hashCode() {
            return _nome.hashCode();
        }

        @Override
        public String toString() {
            return "\"" + getNome() + "\"";
        }

        public String getNome() {
            return _nome;
        }

        protected void setNome(String _nome) {
            this._nome = _nome;
        }

        protected LinkedList<Livro> getLivros() {
            return _livros;
        }

        private String _nome;
        private LinkedList<Livro> _livros;
}
```

```
package pSABbyCRC_UnitTestingSuite;

@SuppressWarnings("serial")
public class AdicionarLivroInexistenteException extends Exception {
     public AdicionarLivroInexistenteException() {}

    public AdicionarLivroInexistenteException(String message)
    {
       super(message);
    }
}
```

```
package pSABbyCRC_UnitTestingSuite;

@SuppressWarnings("serial")
public class BuscaUsuarioComNomeNuloException extends Exception {
     public BuscaUsuarioComNomeNuloException(String message)
    {
       super(message);
    }
}
```

```
package pSABbyCRC_UnitTestingSuite;

@SuppressWarnings("serial")
public class BuscaUsuarioComNomeVazioException extends Exception {
     public BuscaUsuarioComNomeVazioException(String message)
    {
       super(message);
    }
}
```

```
package pSABbyCRC_UnitTestingSuite;

@SuppressWarnings("serial")
public class DevolveLivroDisponivelParaEmprestimoException extends Exception {
     public DevolveLivroDisponivelParaEmprestimoException(String message)
    {
       super(message);
    }
}
```

```
package pSABbyCRC_UnitTestingSuite;

@SuppressWarnings("serial")
public class DevolveLivroNuloParaEmprestimoException extends Exception {
     public DevolveLivroNuloParaEmprestimoException(String message)
    {
       super(message);
    }
}
```

```
package pSABbyCRC_UnitTestingSuite;

@SuppressWarnings("serial")
public class LivroIndisponivelParaEmprestimoException extends Exception {
     public LivroIndisponivelParaEmprestimoException(String message)
    {
       super(message);
    }
}
```

```
package pSABbyCRC_UnitTestingSuite;

@SuppressWarnings("serial")
public class LivroOuUsuarioNulosException extends Exception {
```

```
     public LivroOuUsuarioNulosException(String message)
    {
       super(message);
    }
}
```

```
package pSABbyCRC_UnitTestingSuite;

@SuppressWarnings("serial")
public class TituloOuAutorNuloException extends Exception {
     public TituloOuAutorNuloException(String message)
    {
       super(message);
    }
}
```

```
package pSABbyCRC_UnitTestingSuite;

@SuppressWarnings("serial")
public class TituloOuAutorVazioException extends Exception {
     public TituloOuAutorVazioException(String message)
    {
       super(message);
    }
}
```

```
package pSABbyCRC_UnitTestingSuite;

@SuppressWarnings("serial")
public class UsuarioComNomeVazioException extends Exception {
     public UsuarioComNomeVazioException(String message)
    {
       super(message);
    }
}
```

```
package pSABbyCRC_UnitTestingSuite;

@SuppressWarnings("serial")
public class UsuarioInexistenteException extends Exception {
     public UsuarioInexistenteException(String message)
    {
       super(message);
    }
}
```

```
package pSABbyCRC_UnitTestingSuite;

@SuppressWarnings("serial")
public class UsuarioJaRegistradoException extends Exception {
     public UsuarioJaRegistradoException(String message)
    {
       super(message);
    }
}
-- ---  ----
```

Código de Teste da Versão Preliminar

```
package pSABbyCRC_UnitTestingSuite;

import static org.junit.Assert.assertEquals;

import org.junit.BeforeClass;
import org.junit.Test;

public class BibliotecaTestContextoInicial {
     @BeforeClass
     public static void SetUp() {
          biblioteca = new Biblioteca("ITA");
     }

     /*
      * @Test public void
      * whenSituaçãoInicialEntaoListasLivrosDisponiveisAndEmprestadosSaoVazias()
      * { // T1 // Testa condições de inicio! assertEquals(0,
      * biblioteca.sizeRepositorioLivros()); assertEquals(0,
      * biblioteca.sizeUsuarios()); }
      */

     @Test
     public void whenAdicionoUmLivroEntaoIncrementaTamListaLivrosDisponiveisDeUm()
               throws AdicionarLivroInexistenteException { // T2--T4
          int tam = biblioteca.sizeRepositorioLivros();
          Livro livro1 = new Livro("Java Design Patterns", "Pankaj Kumar"); // T2
          biblioteca.adicionaLivroCatalogo(livro1);
          assertEquals(tam + 1, biblioteca.sizeRepositorioLivros());

          Livro livro2 = new Livro("Clojure", "Sally Fields"); // T3
          biblioteca.adicionaLivroCatalogo(livro2);
          assertEquals(tam + 2, biblioteca.sizeRepositorioLivros());

          Livro livro3 = new Livro("Using CRC Cards", "Nancy Wilkinson"); // T4
          biblioteca.adicionaLivroCatalogo(livro3);
          assertEquals(tam + 3, biblioteca.sizeRepositorioLivros());
     }

     @Test
     public void whenAdicionoOutroLivroComMesmoTituloEAutorEntaoIncrementaTamListaLivrosDisponiveisDeUm()
               throws AdicionarLivroInexistenteException {
          // T5
          int tam = biblioteca.sizeRepositorioLivros();
          Livro livro4 = new Livro("Using CRC Cards", "Nancy Wilkinson");
          biblioteca.adicionaLivroCatalogo(livro4);
          assertEquals(tam + 1, biblioteca.sizeRepositorioLivros());
     }

     @Test(expected = AdicionarLivroInexistenteException.class)
     public void whenAdicionoLivroNuloEntaoAdicionarLivroInexistenteExceptionEhLancada()
               throws AdicionarLivroInexistenteException {
```

```
            // T6
            biblioteca.adicionaLivroCatalogo(null);
      }

      @Test
      public void whenAdicionoUmUsuarioEntaoIncrementaTamListaUsuariosDeUm()
                  throws UsuarioJaRegistradoException, UsuarioComNomeVazioException,
                  UsuarioInexistenteException { // T7--T9
            // T7: Cria 1 usuário novo
            int tam = biblioteca.sizeUsuarios();
            biblioteca.registraUsuario("José");
            assertEquals(tam + 1, biblioteca.sizeUsuarios());

            // T8: Cria segundo usuário novo
            biblioteca.registraUsuario("João");
            assertEquals(tam + 2, biblioteca.sizeUsuarios());

            // T9: Cria terceiro usuário novo
            biblioteca.registraUsuario("Joaquim");
            assertEquals(tam + 3, biblioteca.sizeUsuarios());
      }

      @Test(expected = UsuarioJaRegistradoException.class)
      public void whenAdicionoUsuarioJaExistenteEntaoUsuarioJaRegistradoExceptionEhLancada()
                  throws UsuarioJaRegistradoException, UsuarioComNomeVazioException,
                  UsuarioInexistenteException {
            // T10
            biblioteca.registraUsuario("Joaquim");
            biblioteca.registraUsuario("Joaquim");
      }

      @Test(expected = UsuarioComNomeVazioException.class)
      public void whenAdicionoUsuarioComNomeVazioEntaoUsuarioComNomeVazioExceptionEhLancada()
                  throws UsuarioJaRegistradoException, UsuarioComNomeVazioException,
                  UsuarioInexistenteException {
            // T11
            biblioteca.registraUsuario("");
      }

      @Test(expected = UsuarioInexistenteException.class)
      public void whenAdicionoUsuarioInexistenteEntaoUsuarioInexistenteExceptionEhLancada()
                  throws UsuarioJaRegistradoException, UsuarioComNomeVazioException,
                  UsuarioInexistenteException {
            // T12
            biblioteca.registraUsuario(null);
      }

      private static Biblioteca biblioteca;
}
-- --- ----


package pSABbyCRC_UnitTestingSuite;

import static org.junit.Assert.assertEquals;

import org.junit.Before;
import org.junit.Test;

public class BibliotecaTest {
      @Before
      public void SetUp()
                  throws AdicionarLivroInexistenteException, UsuarioJaRegistradoException, UsuarioComNomeVazioException,
                  UsuarioInexistenteException, BuscaUsuarioComNomeVazioException,
                  BuscaUsuarioComNomeNuloException {
            biblioteca = new Biblioteca("ITA");
            livro1 = new Livro("Java Design Patterns", "Pankaj Kumar");
            biblioteca.adicionaLivroCatalogo(livro1);

            livro2 = new Livro("Clojure", "Sally Fields");
            biblioteca.adicionaLivroCatalogo(livro2);

            livro3 = new Livro("Using CRC Cards", "Nancy Wilkinson");
            biblioteca.adicionaLivroCatalogo(livro3);

            livro4 = new Livro("Using CRC Cards", "Nancy Wilkinson");
            biblioteca.adicionaLivroCatalogo(livro4);

            biblioteca.registraUsuario("José");
            usuario1 = biblioteca.buscaUsuarioPorNome("José");
            biblioteca.registraUsuario("João");
            usuario2 = biblioteca.buscaUsuarioPorNome("João");
            biblioteca.registraUsuario("Joaquim");
            usuario3 = biblioteca.buscaUsuarioPorNome("Joaquim");
      }
      @Test
      public void whenEmprestoUmLivroAUsuarioEntaoLivroFicaIndisponivelParaEmprestimo()
                  throws LivroIndisponivelParaEmprestimoException, LivroOuUsuarioNulosException {
            // T13: Empresta um livro: Livro "Clojure--Sally Fields" para Usuario "José");
            biblioteca.emprestaLivro(livro2, usuario1);
            assertEquals(usuario1, livro2.getUsuario());

            // T14: Empresta outro livro: Livro "Using CRC Cards--Nancy Wilkinson" para Usuario "João");
            biblioteca.emprestaLivro(livro3, usuario2);
            assertEquals(usuario2, livro3.getUsuario());
      }

      @Test
      public void whenDevolvoUmLivroAUsuarioEntaoLivroFicaDisponivelParaEmprestimo()
                  throws LivroIndisponivelParaEmprestimoException, LivroOuUsuarioNulosException,
                  DevolveLivroDisponivelParaEmprestimoException, DevolveLivroNuloParaEmprestimoException {
            // T15: Devolve um livro: Livro "Using CRC Cards--Nancy Wilkinson" de Usuario "João");
            biblioteca.emprestaLivro(livro3, usuario2);
            biblioteca.devolveLivro(livro3);
            assertEquals(null, livro3.getUsuario());
      }

      @Test
      public void whenEmprestoTresLivrosAUmUnicosuarioEntaoListaLivrosdoUsuarioTemTam3()
                  throws LivroIndisponivelParaEmprestimoException,LivroOuUsuarioNulosException {
```

```
        // T16: 3 livros emprestados ao mesmo usuario"
        // 3 Livros: Java Design Patterns--Pankaj Kumar" +
                    // "\n\t\tUsing CRC Cards--Nancy Wilkinson" +
                    // "\n\t\tUsing CRC Cards--Nancy Wilkinson" + "\n\t\tpara Usuario Joaquim");
        biblioteca.emprestaLivro(livro1, usuario3);
        biblioteca.emprestaLivro(livro3, usuario3);
        biblioteca.emprestaLivro(livro4, usuario3);
        assertEquals(3, (usuario3.getLivros()).size());
    }

    @Test(expected = LivroIndisponivelParaEmprestimoException.class)
    public void whenEmprestoLivroJahEmprestadoEntaoLivroDisponivelParaEmprestimoExceptionEhLancada()
            throws LivroIndisponivelParaEmprestimoException, LivroOuUsuarioNulosException {
        // T17: Empresta um livro já emprestado"
        biblioteca.emprestaLivro(livro2, usuario1);
        biblioteca.emprestaLivro(livro1, usuario3);
        biblioteca.emprestaLivro(livro3, usuario3);
        biblioteca.emprestaLivro(livro4, usuario3);

        biblioteca.emprestaLivro(livro2, usuario3);
    }

    @Test(expected = LivroOuUsuarioNulosException.class)
    public void whenEmprestoLivroNuloAUsuarioNaoNuloEntaoLivroOuUsuarioNulosExceptionEhLancada()
            throws LivroIndisponivelParaEmprestimoException, LivroOuUsuarioNulosException {
        // T18a: Empresta livro nulo a usuario não nulo"
        biblioteca.emprestaLivro(null, usuario2);
    }

    @Test(expected = LivroOuUsuarioNulosException.class)
    public void whenEmprestoLivroNaoNuloAUsuarioNuloEntaoLivroOuUsuarioNulosExceptionEhLancada()
            throws LivroIndisponivelParaEmprestimoException, LivroOuUsuarioNulosException {
        // T18b: Empresta livro não nulo a usuario nulo"
        biblioteca.emprestaLivro(livro4, null);
    }

    @Test(expected = LivroOuUsuarioNulosException.class)
    public void whenEmprestoLivroNuloAUsuarioNuloEntaoLivroOuUsuarioNulosExceptionEhLancada()
            throws LivroIndisponivelParaEmprestimoException, LivroOuUsuarioNulosException {
        // T18c: Empresta livro nulo a usuario nulo"
        biblioteca.emprestaLivro(null, null);
    }

    @Test(expected = DevolveLivroDisponivelParaEmprestimoException.class)
    public void whenDevolvoLivroDisponivelParaEmprestimoEntaoDevolveLivroDisponivelParaEmprestimoExceptionEhLancada()
            throws LivroIndisponivelParaEmprestimoException, LivroOuUsuarioNulosException,
            DevolveLivroDisponivelParaEmprestimoException, DevolveLivroNuloParaEmprestimoException {
        // T19: Devolve um livro que está disponível para empréstimo: "Java Design Patterns--Pankaj Kumar"
        biblioteca.emprestaLivro(livro2, usuario1);
        biblioteca.emprestaLivro(livro3, usuario3);
        biblioteca.emprestaLivro(livro4, usuario3);
        biblioteca.devolveLivro(livro1);
    }

    @Test(expected = DevolveLivroNuloParaEmprestimoException.class)
    public void whenDevolvoLivroNuloParaEmprestimoEntaoDevolveLivroNuloParaEmprestimoExceptionEhLancada()
            throws LivroIndisponivelParaEmprestimoException, LivroOuUsuarioNulosException,
            DevolveLivroDisponivelParaEmprestimoException, DevolveLivroNuloParaEmprestimoException {
        // T20: Devolve um livro nulo"
        biblioteca.devolveLivro(null);
    }

    @Test
    public void whenBuscoLivroPeloNrCatalogoEntaoRetornoLivroCujoNrCatalogoConfere() {
        // T21: Busca livro por NrCatalogo existente
        // Livro 2: "Clojure--Sally Fields"
        Livro livro = biblioteca.buscaLivroPorNrCatalogo(2);
        assertEquals(2, livro.getNrCatalogo());
    }

    @Test
    public void whenBuscoLivroPeloNrCatalogoInexistenteEntaoLivroEhNulo() {
        // T22: Busca livro por NrCatalogo inexistente: 0 e 5
        Livro livroHum = biblioteca.buscaLivroPorNrCatalogo(0);
        Livro livroDois = biblioteca.buscaLivroPorNrCatalogo(5);
        assertEquals(null, livroHum);
        assertEquals(null, livroDois);
    }

    @Test
    public void whenBuscoLivroPorTituloEAutorEntaoRetornoLivroCujoTituloEAutorConfere()
            throws TituloOuAutorVazioException, TituloOuAutorNuloException {
        // T23: Busca livro por Titulo e Autor existente: livros 1 e 3
        Livro livroTres = biblioteca.buscaLivroPorTituloAutor("Using CRC Cards", "Nancy Wilkinson");
        assertEquals(3, livroTres.getNrCatalogo());

        Livro livroHum = biblioteca.buscaLivroPorTituloAutor("Java Design Patterns", "Pankaj Kumar");
        assertEquals(1, livroHum.getNrCatalogo());
    }

    @Test
    public void whenBuscoLivroPorTituloOuAutorENaoEncontroEntaoRetornoLivroNulo()
            throws TituloOuAutorVazioException, TituloOuAutorNuloException {
        // T24: Busca livro por Titulo e Autor e não encontra
        Livro livro = biblioteca.buscaLivroPorTituloAutor("Using CRC Cards", "Pankaj Kumar");
        assertEquals(null, livro);

        livro = biblioteca.buscaLivroPorTituloAutor("Padrões de Projeto em Java", "Pankaj Kumar");
        assertEquals(null, livro);

        livro = biblioteca.buscaLivroPorTituloAutor("Padrões de Projeto em Java", "Eduardo Guerra");
        assertEquals(null, livro);
    }

    @Test(expected = TituloOuAutorVazioException.class)
    public void whenBuscoLivroPorTituloNaoVazioEAutorVazioEntaoTituloOuAutorVazioExceptionEhLancada()
            throws TituloOuAutorVazioException, TituloOuAutorNuloException {
        // T25a: Busca livro por Titulo não vazio e Autor vazio
        biblioteca.buscaLivroPorTituloAutor("Using CRC Cards", "");
    }
}
```

```
	@Test(expected = TituloOuAutorVazioException.class)
	public void whenBuscoLivroPorTituloVazioEAutorNaoVazioEntaoTituloOuAutorVazioExceptionEhLancada()
			throws TituloOuAutorVazioException, TituloOuAutorNuloException {
		// T25b: Busca livro por Titulo vazio e Autor não vazio
		biblioteca.buscaLivroPorTituloAutor("", "Nancy Wilkinson");
	}

	@Test(expected = TituloOuAutorVazioException.class)
	public void whenBuscoLivroPorTituloVazioEAutorVazioEntaoTituloOuAutorVazioExceptionEhLancada()
			throws TituloOuAutorVazioException, TituloOuAutorNuloException {
		// T25c: Busca livro por Titulo vazio e Autor vazio
		biblioteca.buscaLivroPorTituloAutor("", "");
	}

	@Test(expected = TituloOuAutorNuloException.class)
	public void whenBuscoLivroPorTituloNaoNuloEAutorNuloEntaoTituloOuAutorVazioExceptionEhLancada()
			throws TituloOuAutorVazioException, TituloOuAutorNuloException {
		// T26a: Busca livro por Titulo não nulo e Autor nulo
		biblioteca.buscaLivroPorTituloAutor("Using CRC Cards", null);
	}

	@Test(expected = TituloOuAutorNuloException.class)
	public void whenBuscoLivroPorTituloNuloEAutorNaoNuloEntaoTituloOuAutorVazioExceptionEhLancada()
			throws TituloOuAutorVazioException, TituloOuAutorNuloException {
		// T26b: Busca livro por Titulo nulo e Autor não nulo
		biblioteca.buscaLivroPorTituloAutor(null, "Nancy Wilkinson");
	}

	@Test(expected = TituloOuAutorNuloException.class)
	public void whenBuscoLivroPorTituloNuloEAutorNuloEntaoTituloOuAutorVazioExceptionEhLancada()
			throws TituloOuAutorVazioException, TituloOuAutorNuloException {
		// T26c: Busca livro por Titulo nulo e Autor nulo
		biblioteca.buscaLivroPorTituloAutor(null, null);
	}

	@Test
	public void whenBuscoUsuarioPorNomeExistenteEntaoRetornoUsuarioValido()
			throws BuscaUsuarioComNomeVazioException, BuscaUsuarioComNomeNuloException {
		// T27: Busca usuario por Nome existente
		Usuario usuario = biblioteca.buscaUsuarioPorNome("José");
		assertEquals("José", usuario.getNome());

		usuario = biblioteca.buscaUsuarioPorNome("Joaquim");
		assertEquals("Joaquim", usuario.getNome());
	}

	@Test
	public void whenBuscoUsuarioPorNomeInexistenteEntaoRetornoUsuarioNulo()
			throws BuscaUsuarioComNomeVazioException, BuscaUsuarioComNomeNuloException {
		// T28: Busca usuario por Nome inexistente
		Usuario usuario = biblioteca.buscaUsuarioPorNome("Eduardo");
		assertEquals(null, usuario);

		usuario = biblioteca.buscaUsuarioPorNome("Clovis");
		assertEquals(null, usuario);
	}

	@Test(expected = BuscaUsuarioComNomeVazioException.class)
	public void whenBuscoUsuarioPorNomeVazioEntaoBuscaUsuarioComNomeVazioExceptionEhLancada()
			throws BuscaUsuarioComNomeVazioException, BuscaUsuarioComNomeNuloException {
		// T29: Busca usuario por Nome vazio
		biblioteca.buscaUsuarioPorNome("");
	}

	@Test(expected = BuscaUsuarioComNomeNuloException.class)
	public void whenBuscoUsuarioPorNomeNuloEntaoBuscaUsuarioComNomeNuloExceptionEhLancada()
			throws BuscaUsuarioComNomeVazioException, BuscaUsuarioComNomeNuloException {
		// T29: Busca usuario por Nome vazio
		biblioteca.buscaUsuarioPorNome(null);
	}

	private Biblioteca biblioteca;
	private Livro livro1, livro2, livro3, livro4;
	private Usuario usuario1, usuario2, usuario3;
}
-- --- ----

package pSABbyCRC_UnitTestingSuite;

import static org.junit.Assert.assertEquals;

import org.junit.Before;
import org.junit.Test;

public class LivroTest {
	@Before
	public void SetUp() {
		livro = new Livro("Java Design Patterns", "Pankaj Kumar");
	}

	@Test
	public void whenSituaçãoInicialEntaoLivroDisponivelParaEmprestimo() {
		// T1: Testa condições de inicio!
		// Assumo que Biblioteca não deixa criar livro com
		// titulo e/ou autor com nomes vazios ou nulos!
		assertEquals(null, livro.getUsuario());
		assertEquals("Java Design Patterns", livro.getTitulo());
		assertEquals("Pankaj Kumar", livro.getAutor());
	}

	@Test
	public void whenAnexaUsuarioNaoNuloAoLivroEntaoLivroFicaIndisponivelParaEmprestimo() {
		// T2: Anexa usuario não nulo
		Usuario usuario1 = new Usuario("José");
		livro.anexaUsuarioAoLivro(usuario1);
		assertEquals(usuario1, livro.getUsuario());
	}

	@Test
	public void whenAnexaUsuarioNuloAoLivroEntaoLivroFicaDisponivelParaEmprestimo() {
		// T3: Anexa usuario nulo
```

```
		Usuario usuario1 = null;
		livro.anexaUsuarioAoLivro(usuario1);
		assertEquals(usuario1, livro.getUsuario());
	}

	@Test
	public void whenDesanexaUsuarioNaoNuloDoLivroEntaoLivroFicaDisponivelParaEmprestimo() {
		// T4: Desanexa usuario não nulo
		Usuario usuario1 = new Usuario("José");
		livro.anexaUsuarioAoLivro(usuario1);
		livro.desanexaUsuarioDoLivro();
		assertEquals(null, livro.getUsuario());
	}

	@Test
	public void whenDesanexaUsuarioNuloDoLivroEntaoLivroContinuaDisponivelParaEmprestimo() {
		// T5: Desanexa usuario nulo
		Usuario usuario1 = null;
		livro.anexaUsuarioAoLivro(usuario1);
		livro.desanexaUsuarioDoLivro();
		assertEquals(null, livro.getUsuario());
	}

	private Livro livro;
}
-- ---  ----

package pSABbyCRC_UnitTestingSuite;

import java.util.Iterator;
import java.util.LinkedList;

public class Usuario implements Comparable<Object> {
	public Usuario(String nome) {
		setNome(nome);
		_livros = new LinkedList<Livro>();
	}

	public void anexaLivroAoUsuario(Livro livro) {
		if (livro != null)
			_livros.add(livro);
	}

	public void desanexaLivroDoUsuario(Livro livro) {
		//if (livro != null)
			_livros.remove(livro);
	}

	public void exibe() {
		System.out.println("\t\t" + "Nome: " + "\t\t" + getNome());
		this.exibeLivrosUsuario();
	}

	private void exibeLivrosUsuario() {
		System.out.println("\t\t" + "\\//Livros emprestados:");
		if (_livros.size() != 0) {
			Iterator<Livro> iter = _livros.iterator();
			while (iter.hasNext() == true) {
				Livro livro = (Livro) iter.next();
				System.out.println("\t\t\t" + livro.getNrCatalogo() + " "
						+ livro);
			}
		} else
			System.out.println("\t\t" + "---> Nenhum livro emprestado");
		System.out.println("\t\t" + "\\//\\//\\//");
		System.out.println();
	}

	@Override
	public boolean equals(Object obj) {
		return this.compareTo(obj) == 0;

	}

	@Override
	public int compareTo(Object obj) {
		Usuario usuario = (Usuario) obj;
		String nome = usuario.getNome();
		return _nome.compareTo(nome);
	}

	@Override
	public int hashCode() {
		return _nome.hashCode();
	}

	@Override
	public String toString() {
		return "\"" + getNome() + "\"";
	}

	public String getNome() {
		return _nome;
	}

	protected void setNome(String _nome) {
		this._nome = _nome;
	}

	protected LinkedList<Livro> getLivros() {
		return _livros;
	}

	private String _nome;
	private LinkedList<Livro> _livros;
}
```