A)

Data Class: classes que não definem funcionalidades próprias.

```
package pSABbyCRC_UnitTestingSuite;

public class CentralNumeracaoLivro {
	private int _nrUnico = 0;
}
```

Comments: métodos com muitos comentários desnecessários e confusos

```
	public Usuario getUsuario() {
		// Responsabilidade: sabe disponibilidade de emprŽstimo:
		// _usuario = null --> livro dispon'vel para emprŽstimo
		// _usu‡rio = objeto Usuario --> livro indispon'vel para
emprŽstimo
		return _usuario;
	}
```

God Class: classes com alta responsabilidade no sistema.

```
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(
					"--->N<o 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("--->J‡ existe usu‡rio com o nome \""
							+ nome + "\"! Use outro
nome!");
			} else
				throw new UsuarioComNomeVazioException("--->N<o
pode registrar usuario com nome vazio!");
		} else
			throw new UsuarioInexistenteException("--->N<o pode
registrar usuario inexistente!");
	}

	public void emprestaLivro(Livro livro, Usuario usuario)
			throws LivroIndisponivelParaEmprestimoException,
			LivroOuUsuarioNulosException {
		if ((livro == null) && (usuario == null))
			throw new LivroOuUsuarioNulosException(
					"--->Livro e Usu‡rio inexistentes!");
		if (livro != null) {
			if (usuario != null) {
				if (livro.getUsuario() == null) {
					usuario.anexaLivroAoUsuario(livro);
					livro.anexaUsuarioAoLivro(usuario);
				} else
					throw new
LivroIndisponivelParaEmprestimoException(
							"--->Livro " + livro
								+ " indispon'vel
para emprŽstimo!");
			} else
				throw new LivroOuUsuarioNulosException(
					"--->N<o pode emprestar livro a
```

```
Usu‡rio inexistente!");
            } else
                  throw new LivroOuUsuarioNulosException(
                              "--->N<o 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 est‡ dispon'vel
para emprŽstimo!");
            } else
                  throw new DevolveLivroNuloParaEmprestimoException(
                              "--->N<o pode emprestar livro
inexistente!");
      }

      public Livro buscaLivroPorNrCatalogo(int nrUnico) {
            // nrUnico <= zero devolve nulo: n<o 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
Ž(s<o) vazio(s)<<<");
            } else
                  throw new TituloOuAutorNuloException(
                            "--->Nome do titulo e/ou do autor Ž(s<o)
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 usu‡rio Ž vazio<<<");
            } else
                  throw new BuscaUsuarioComNomeNuloException(
                              "--->Nome do usu‡rio Ž nulo<<<");
```

```
            return usuarioAchado;
      }

      public void exibeLivrosDisponiveis() {
            System.out.println("Biblioteca: " + _nome);
            System.out.println(">>>Livros Dispon'veis para
EmprŽstimo<<<");
            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
reposit--rio");
            System.out.println("<<< Livros Dispon'veis >>>");
            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
reposit--rio");
            System.out.println("<<< Livros Emprestados >>>");
            System.out.println();
      }

      public void exibeUsuarios() {
            System.out.println("Biblioteca: " + _nome);
```

```
            System.out.println(">>>Usu‡rios 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 usu‡rio na
Biblioteca");
            System.out.println("<<< Usu‡rios >>>");
            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;
}
```

God Package: pacotes grandes, que possuem muitos dependentes

Long Method

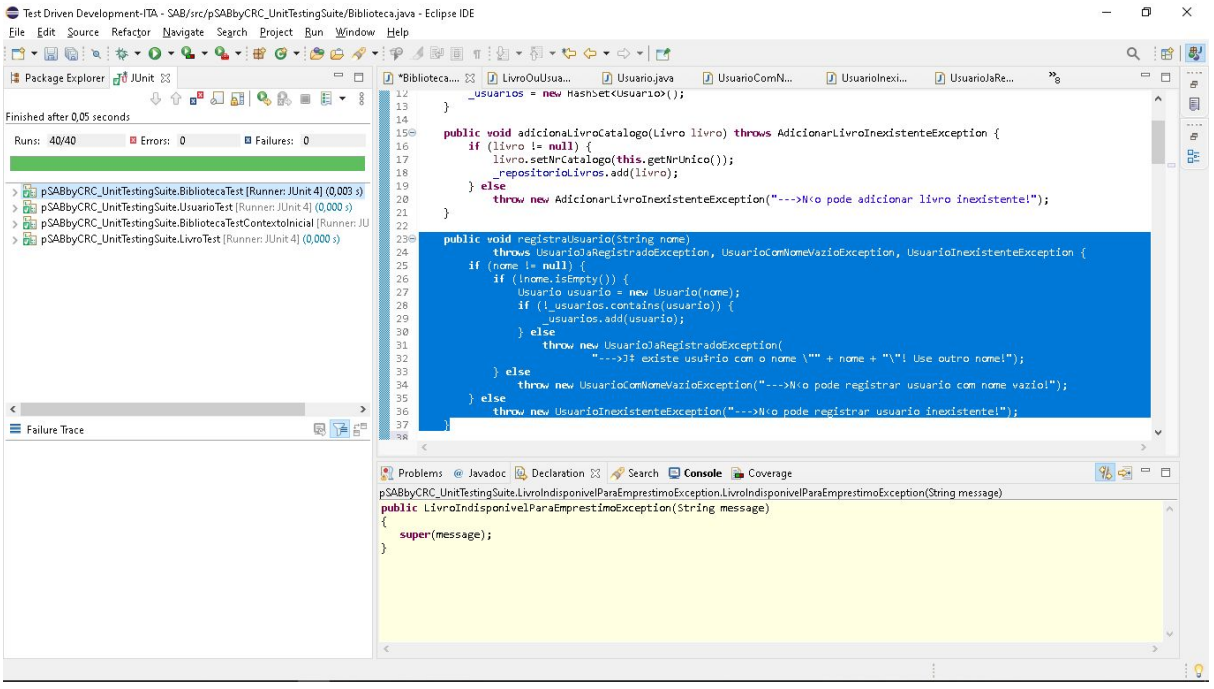
```
public void emprestaLivro(Livro livro, Usuario usuario)
                  throws LivroIndisponivelParaEmprestimoException,
                  LivroOuUsuarioNulosException {
            if ((livro == null) && (usuario == null))
                  throw new LivroOuUsuarioNulosException(
                              "--->Livro e Usu‡rio inexistentes!");
            if (livro != null) {
                  if (usuario != null) {
                        if (livro.getUsuario() == null) {
                              usuario.anexaLivroAoUsuario(livro);
                              livro.anexaUsuarioAoLivro(usuario);
                        } else
                              throw new
LivroIndisponivelParaEmprestimoException(
                                          "--->Livro " + livro
                                                      + " indispon'vel
para emprŽstimo!");
                  } else
                        throw new LivroOuUsuarioNulosException(
                                    "--->N<o pode emprestar livro a
Usu‡rio inexistente!");
            } else
                  throw new LivroOuUsuarioNulosException(
                              "--->N<o pode emprestar livro
inexistente!");
      }
```

b) c)

1.1 Código antes da refatoração

```
      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(
                                          "--->J‡ existe usu‡rio com o
nome \"" + nome + "\"! Use outro nome!");
                  } else
                        throw new UsuarioComNomeVazioException("--->N<o
pode registrar usuario com nome vazio!");
            } else
                  throw new UsuarioInexistenteException("--->N<o pode
registrar usuario inexistente!");
      }
```

1.2- Testes

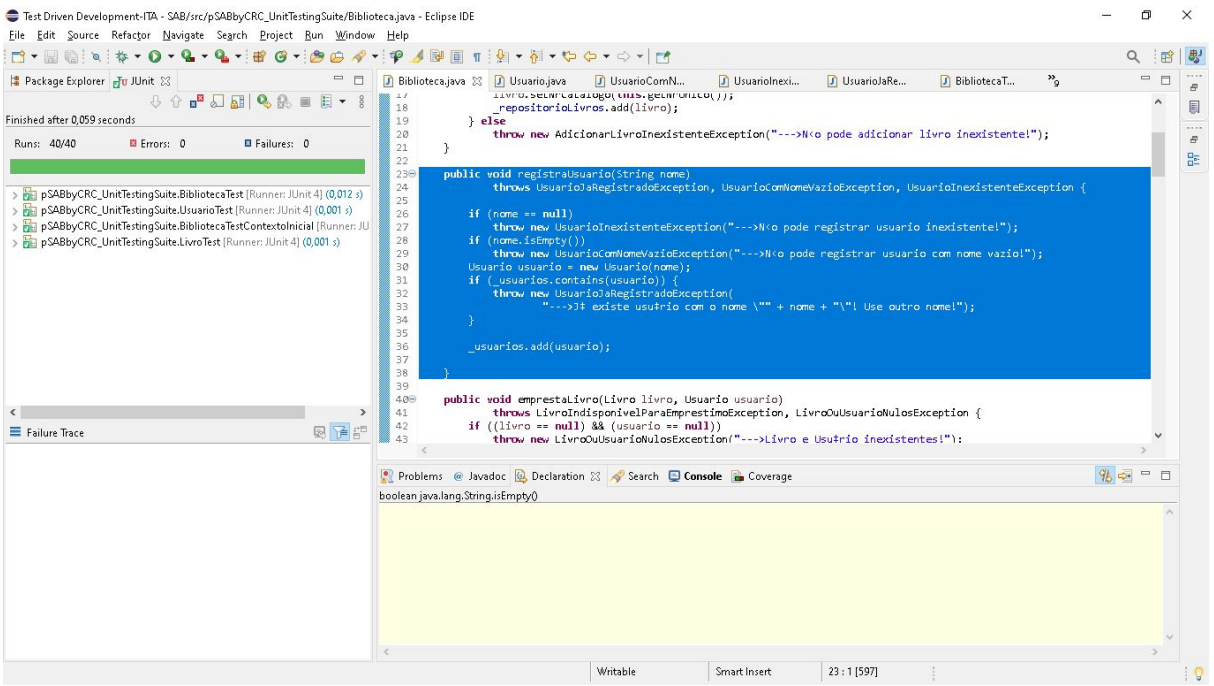


1.3 if, elses aninhados

1.4 Refatoraçãp

```
      public void registraUsuario(String nome)
                  throws UsuarioJaRegistradoException,
UsuarioComNomeVazioException, UsuarioInexistenteException {

            if (nome == null)
                  throw new UsuarioInexistenteException("--->N<o pode
registrar usuario inexistente!");
            if (nome.isEmpty())
                  throw new UsuarioComNomeVazioException("--->N<o pode
registrar usuario com nome vazio!");
            Usuario usuario = new Usuario(nome);
            if (_usuarios.contains(usuario)) {
                  throw new UsuarioJaRegistradoException(
                              "--->J‡ existe usu‡rio com o nome \"" +
nome + "\"! Use outro nome!");
            }

            _usuarios.add(usuario);

      }
```

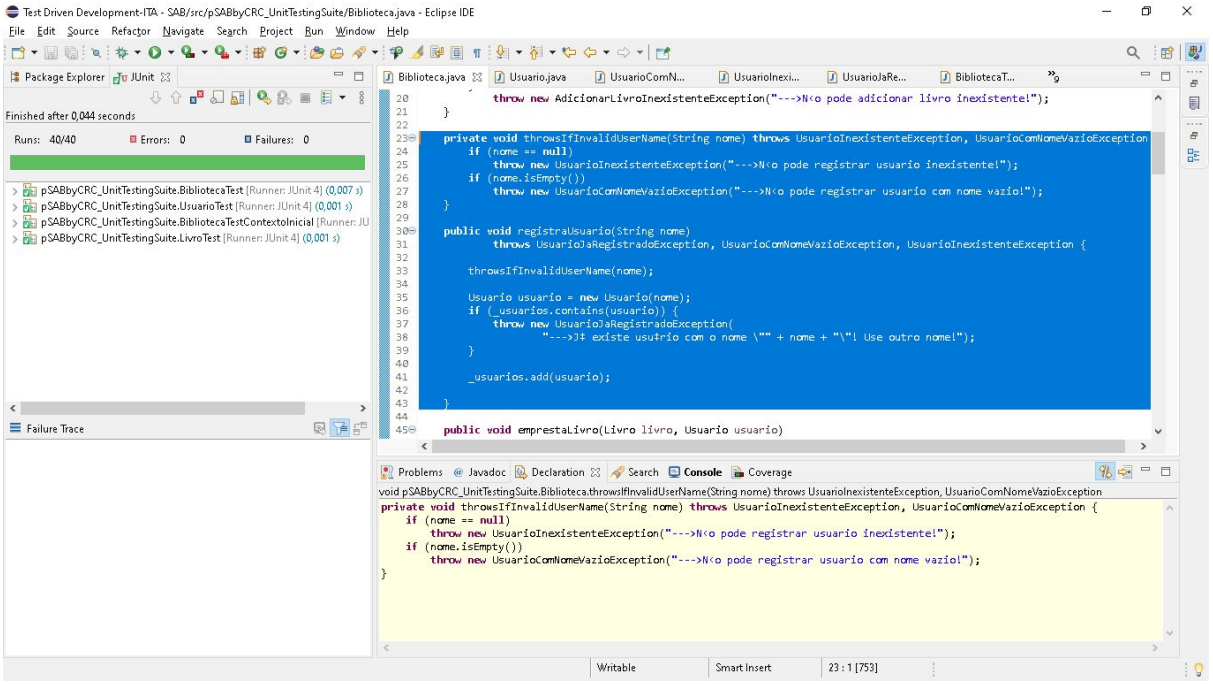
1.5- Testes passando



1.6-Extract Method

```
      private void throwsIfInvalidUserName(String nome) throws
UsuarioInexistenteException, UsuarioComNomeVazioException {
            if (nome == null)
                  throw new UsuarioInexistenteException("--->N<o pode
registrar usuario inexistente!");
            if (nome.isEmpty())
                  throw new UsuarioComNomeVazioException("--->N<o pode
registrar usuario com nome vazio!");
      }

      public void registraUsuario(String nome)
                  throws UsuarioJaRegistradoException,
UsuarioComNomeVazioException, UsuarioInexistenteException {

            throwsIfInvalidUserName(nome);

            Usuario usuario = new Usuario(nome);
            if (_usuarios.contains(usuario)) {
                  throw new UsuarioJaRegistradoException(
                              "--->J‡ existe usu‡rio com o nome \"" +
nome + "\"! Use outro nome!");
            }

            _usuarios.add(usuario);

      }
```

1.7 - Testes passando

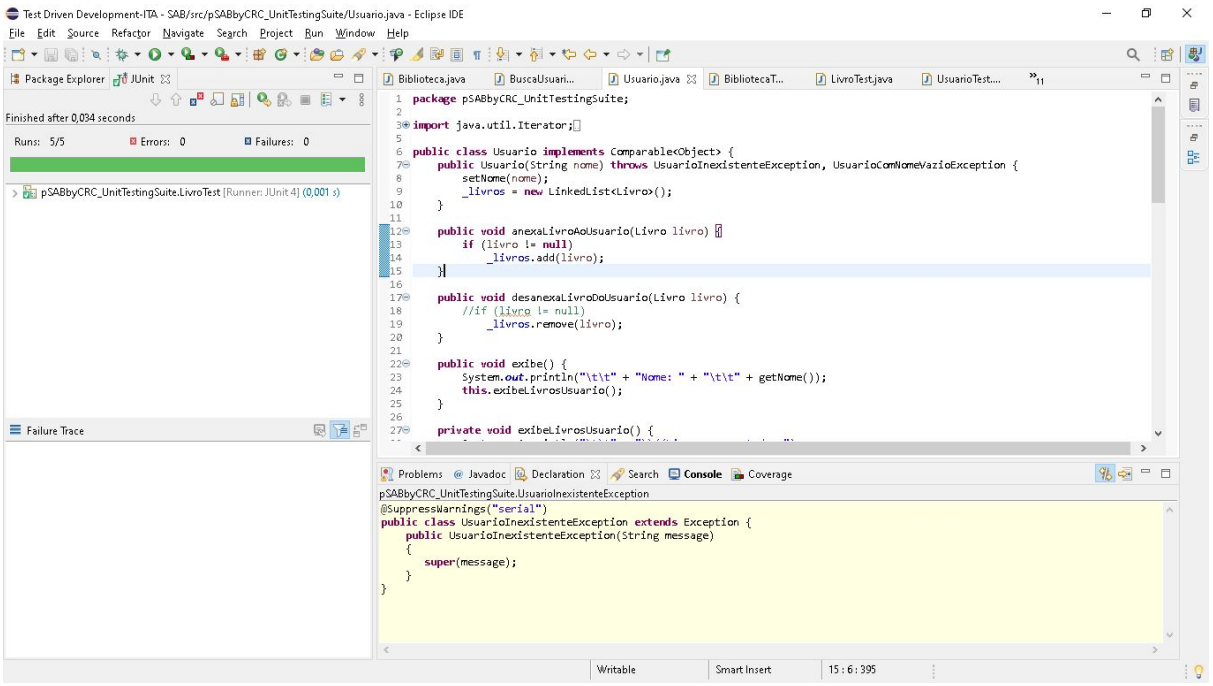


1.8

Movendo throwsIfInvalidName para classe usuário

```
	private void throwsIfInvalidUserName(String nome) throws
UsuarioInexistenteException, UsuarioComNomeVazioException {
		if (nome == null)
			throw new UsuarioInexistenteException("--->N<o pode
registrar usuario inexistente!");
		if (nome.isEmpty())
			throw new UsuarioComNomeVazioException("--->N<o pode
registrar usuario com nome vazio!");
	}

	protected void setNome(String nome) throws
UsuarioInexistenteException, UsuarioComNomeVazioException {
		throwsIfInvalidUserName(nome);
		this._nome = nome;
	}
```

1.9 - Testes passando



1.10 - Extraindo throwsIfUser Exists

```
	private void throwsIfUserExists(Usuario usuario) throws
UsuarioJaRegistradoException {
		if (_usuarios.contains(usuario)) {
			throw new UsuarioJaRegistradoException(
					"--->J‡ existe usu‡rio com o nome \"" +
usuario.getNome() + "\"! Use outro nome!");
		}
	}

	public void registraUsuario(String nome)
			throws UsuarioJaRegistradoException,
UsuarioComNomeVazioException, UsuarioInexistenteException {

		Usuario usuario = new Usuario(nome);

		throwsIfUserExists(usuario);

		_usuarios.add(usuario);

	}
```

1.11 Testes passando