# INSTITUTO TECNOLÓGICO DE AERONÁUTICA (ITA)

## TDD – Desenvolvimento de Software Guiado por Testes

## Semana 02
## Análise e Refatoração de Métodos do SAB (Sistema Automatizado de Bibliotecas)

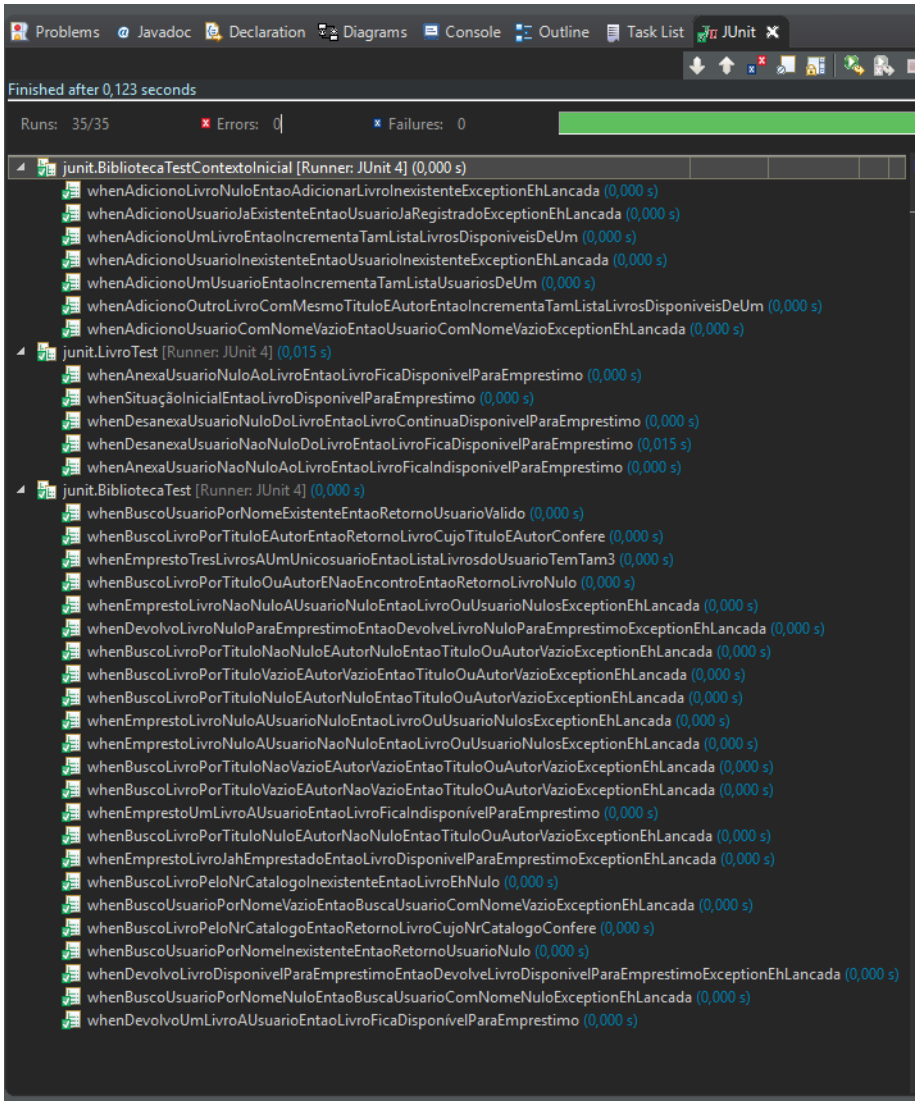
Coursera / Aluno:
Vagner Panarello Filho

São Paulo, 3 de junho de 2016

# Índice

# 1. Executando testes do projeto SAB original.

Após a cópia de todo o conteúdo do projeto SAB em suas devidas classes dentro do ambiente Eclipse. Foi executada as 3 classes de testes do Junit. A lista abaixo informa que dos 35 testes inicialmente implementados estão com status “verde”. Passaram sem que fosse feita nenhuma modificação.



## 2. Identificando pontos a serem fatorados “mau cheiros”.

Abaixo o código original do projeto SAB o método “registraUsuario” foram encontrados 2 mau cheiros mais evidentes. Indicados com setas coloridas, a descrição de cada um deles está mais abaixo a imagem.

```
    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
```

Utilizando expressões booleanas negativas “!” nos testes “IF”

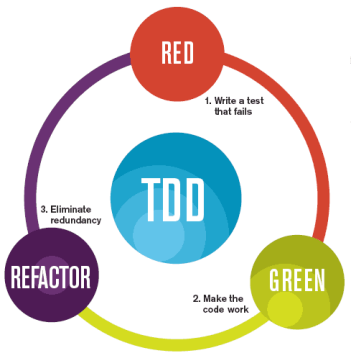
“IFs” aninhados com IF-ELSE

Começo de variáveis de instância com “underline” (_).

* Apesar de não mensionado no escopo do curso a prática mais formal para sinalizar variáveis de instância é usar o operador “this”.

Mensagens de exceção atrapalham a legibilidade do código.

## 3. Ciclos de refatoração.



## a) Renomear variável de instância

Decidi substituir o “underline” (_) das variáveis de instância pelo operador “this” pois além de sinalizar que se trata efetivamente de uma variável de instância, ainda mantem o código seguro por forçar o compilador a referenciar essa variável aos atributos da classe e não, acidentalmente, uma variável do método que tenha o mesmo nome.

Após a efetuar esse mudança foram executados os testes que passaram sem problemas.

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

    if (nome != null) {
        if (!nome.isEmpty()) {
            Usuario usuario = new Usuario(nome);
            if (!this.usuarios.contains(usuario)) {
                this.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!");
}
```

## b) Modificar expressões booleanas para não serem negativas

Para está modificação basicamente foram trocadas dentro das condições IF-ELSE as operações que pertenciam ao ELSE foram para o IF e as do IF para o ELSE. Logo após isso foi retirado o sinal de negação “!” das usas respectivas condições.

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

## c) Cláusula de guarda

Como a cada exceção que é lançada o método não segue sua rota normal de execução, a utilização das cláusulas de guarda simplificam substancialmente o método. Pois nesses casos é possível elminar os “ELSE” mantendo um só fluxo.

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

### d) Otimização das mensagens nas exceções

Por último optei por passar as string de mensagens para uma Array de strings para manter o código mais legível.

```
public void registraUsuario(String nome) throws UsuarioJaRegistradoException,
                                                UsuarioComNomeVazioException,
                                                UsuarioInexistenteException
{
    String[] ExceptionsMsg = { "--->Não pode registrar usuario inexistente!",
                               "--->Não pode registrar usuario com nome vazio!",
                               "--->Já existe usuário com o nome \"" + nome + "\"! Use outro nome!" };

    if (nome == null) throw new UsuarioInexistenteException(ExceptionsMsg[0]);
    if (nome.isEmpty()) throw new UsuarioComNomeVazioException(ExceptionsMsg[1]);
    Usuario usuario = new Usuario(nome);
    if (this.usuarios.contains(usuario)) throw new UsuarioJaRegistradoException(ExceptionsMsg[2]);
    this.usuarios.add(usuario);
}
```

## 4. Últimas considerações

Os ciclos de refatoração foram respeitados em todo o processo. Com uma modificação seguido de teste. Após validado passava-se para a próxima modificação e assim por diante. Até o término de todas as refatorações.