Universidade Federal do Ceará
Departamento de Computação
Curso de Ciência da Computação

Disciplina	Técnicas de Programação II - CK0129	**Semestre**	2015/2
Professor	Lincoln Souza Rocha		

Prática de Laboratório 02 – Trabalhando com Git/GitHub em Equipe

Essa Prática de Laboratório tem por objetivo exercitar os conceitos sobre Git/GitHub e trabalho colaborativo de desenvolvimento de software. Para isso, o líder da sua equipe de fazer *download* do projeto do Sistema Bancário, disponível na página da disciplina no SIGAA, e seguir as instruções dadas abaixo. A equipe deve estabelecer um fluxo de trabalho com o Git/GitHub, como o descrito abaixo:

Passo 1) Começando os trabalhos

```
$ git checkout master (ativando no ramo master)
$ git pull (recuperando a última versão do repositório remoto da equipe)
```

Passo 2) Criando um ramo para armazenar as alterações

```
$ git checkout –b <work_branch> (criando o ramo onde serão feitas as alterações locais)
```

Passo 3) Recuperando as alterações realizadas desde que comecei a trabalhar

```
$ git checkout master (ativando no ramo master)
$ git pull (recuperando a última versão do repositório remoto da equipe)
```

Passo 4) Sincronizando a base do meu ramo de trabalho com as alterações remotas

```
$ git checkout <work_branch> (ativando no ramo <work_branch>)
$ git rebase master (atualizando a base do ramo <work_branch> com a do master)
```

Passo 5) Mesclando as minhas alterações (lembre-se de tratar os conflitos, caso existam)

```
$ git checkout master (ativando no ramo master)
$ git merge <work_branch> (fazendo o merge dos ramos master e <work_branch>)
```

Passo 6) Finalmente, enviando minhas alterações para o repositório remoto

```
$ git push origin máster (enviando as alterações para o servidor remoto)
```

Tarefa 1) Setup – o líder deve colocar o código do Sistema Bancário sob controle de versão no repositório criado para sua equipe na organização da disciplina no GitHub (https://github.com/orgs/TPII20152). Certifique-se que todos os membros da sua equipe estão cadastrados na organização da disciplina e pertencem a equipe correta. Em seguida, você e sua equipe devem distribuir as tarefas, descritas abaixo, a serem feitas entre todos os membros da equipe. Utilize o gerenciador de Issues do GitHub para documentar as tarefas, atribuir responsáveis e acompanhar o status do andamento.

Tarefa 2) Bug Fix – O método `debitar(double valor)` da classe `ContaImposto` não está observando o caso excepcional de saldo insuficiente. Corrija esse bug!

Tarefa 3) Bug Fix - A classe `ArrayContas` possui limite fixo de 100 posições para armazenar contas. Ela deve suportar tamanho ilimitado. Corrija esse bug!

Tarefa 4) New Feature – Implementar persistência em arquivo. Para isso você deve implementar uma classe que estende `IRepositorioContas`. Você pode usar a API padrão do Java ou algum *framework* como o *XStream* (http://x-stream.github.io/).

Tarefa 5) New Feature – Implementar uma GUI com Java Swing para o Sistema Bancário (Terminal de Autoatendimento).

Tarefa 6) Construa casos de teste JUnit para cada uma das unidades do Sistema Bancário que atendam aos Cenários de Teste especificados. Desse modo, para cada unidade deve ser criado um Caso de Teste e para cada Cenário de Teste deve ser implementado em um método específico do Caso de Teste. (OBS. sinta-se livre para propor mais Cenários de Teste para os Casos de Teste sugeridos nessa prática.)

1) Caso de Teste da classe `Conta` (`ContaTest`):
    1. Cenário de Teste - Creditar Normal (`testCreditarNormal`): crie um teste que verifique se a operação `creditar()` está funcionando corretamente, para isso, escolha como entrada um valor positivo, execute a operação `creditar()` e verifique se o saldo da conta foi acrescido do valor esperado;
    2. Cenário de Teste - Creditar Negativo (`testCreditarNegativo`): crie um teste que verifique se a operação `creditar()` está funcionando corretamente, para isso, escolha como entrada um valor negativo, execute a operação `creditar()` e verifique se o saldo resultante da conta permanece o mesmo de antes da execução da operação;
    3. Cenário de Teste - Debitar Normal (`testDebitarNormal`): crie um teste que verifique se a operação `debitar()` está funcionando corretamente, para isso, escolha como entrada um valor positivo, execute a operação `debitar()` e verifique se o saldo da conta foi decrescido do valor esperado;
    4. Cenário de Teste - Debitar Negativo (`testDebitarNegativo`): crie um teste que verifique se a operação `debitar()` está funcionando corretamente, para isso, escolha como entrada um valor negativo, execute a operação `debitar()` e verifique se o saldo resultante da conta permanece o mesmo de antes da execução da operação.

2) Caso de Teste da classe `ContaEspecial` (`ContaEspecialTest`):
    1. Cenário de Teste - Creditar Normal (`testCreditarNormal`): crie um teste que verifique se a operação `creditar()` está funcionando corretamente, para isso, escolha como entrada um valor positivo, execute a operação `creditar()` e verifique se o saldo da conta foi acrescido do valor esperado;
    2. Cenário de Teste - Creditar Negativo (`testCreditarNegativo`): crie um teste que verifique se a operação `creditar()` está funcionando corretamente, para isso, escolha como entrada um valor negativo, execute a operação `creditar()` e verifique se o saldo resultante da conta permanece o mesmo de antes da execução da operação;
    3. Cenário de Teste - Bônus Normal (`testBonusNormal`): crie um teste que verifique se a operação de `creditar()` está incrementando o bônus corretamente, para isso, escolha como entrada um valor positivo, execute a operação `creditar()` e verifique se o bônus da conta foi acrescido do valor esperado (i.e., acrescido de 0,01 do valor creditado);
    4. Cenário de Teste - Bônus Negativo (`testBonusNormal`): crie um teste que verifique se a operação de `creditar()` está incrementando o bônus corretamente, para isso, escolha como entrada um valor negativo, execute a operação `creditar()` e verifique se o bônus da conta permanece o mesmo de antes da execução da operação;
    5. Cenário de Teste – Render Bônus (`testRendeBonus`): crie um teste que verifique se a operação de `rendeBonus()` está rendendo o bônus corretamente, para isso, realize operações de créditona conta a fim de incrementar o valor dos bônus, que é incrementado em 0,01 do valor creditado em cada operação de crédito. Após isso, verifique a quantidade de bônus acumulados (`obterBonus()`) e, em seguida,

execute a operação `rendeBonus()` e verifique se o saldo da conta foi acrescido da quantidade de bônus acumulada.

3) Caso de Teste da classe `ContaPoupanca` (`ContaPoupancaTest`):
   1. Cenário de Teste – Render Juros Normal (`testRendeJuros`): crie um teste que verifique se a operação de `rendeJuros()` está rendendo o juros corretamente, para isso, realize operações de crédito na conta a fim de incrementar o valor do saldo e, em seguida, escolha como taxa de juros um valor positivo (e.g., 0,01), execute a operação `rendeJuros()` com essa taxa e verifique se o saldo da conta foi acrescido do valor esperado (i.e., do produto da taxa de juros utilizada multiplicada pelo saldo anterior).

4) Caso de Teste da classe `ContaImposto` (`ContaImpostoTest`):
   1. Cenário de Teste – Debitar Normal (`testDebitarNormal`): crie um teste que verifique se a operação `debitar()` está funcionando corretamente, para isso, escolha como entrada um valor positivo, execute a operação `debitar()` e verifique se o saldo da conta foi decrescido do valor esperado (i.e., do valor mais o acréscimo de imposto que é valor debitado multiplicado por 0.001);
   2. Cenário de Teste – Debitar Negativo (`testDebitarNegativo`): crie um teste que verifique se a operação `debitar()` está funcionando corretamente, para isso, escolha como entrada um valor negativo, execute a operação `debitar()` e verifique se o saldo resultante da conta permanece o mesmo de antes da execução da operação.

Tarefa 7) Com base nos Casos de Teste e Cenários de Testes descritos acima, especifique e implemente novos Casos de Teste e Cenários de Teste para as classes: `BancoBrasil`, `VectorContas`, `ArrayContas` e demais classes de persistência que forem implementadas. Para lidar com testes onde exceções são esperadas use a anotação `@Test(expected = <NomeDaExceção>.class)`.