PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Curso de Bacharelado em Engenharia de Software

Disciplina	Departamento	Turno	Período
Programação Modular	Ciência da Computação	Manhã/Noite	2°

Professor
Hugo de Paula (hugo@pucminas.br)

Matrícula:	Aluno:

Lista de Exercícios 2
Atributos estáticos, destrutores e TDD

1. Considere o código abaixo:

```
public class Instanciadora
{
        private static int id = 0;
        private String tipoInstancia;

        public Instanciadora(String tipoRequerido)
        {
                id = id + 1;
                tipoInstancia = tipoRequerido;
        }

        public String toString()
        {
                return "Tipo: " + tipoInstancia + ", Id: " + id + "; ";
        }

        public static void main(String [] args)
        {
                Instanciadora[] instancias = new Instanciadora[2];
                instancias[0] = new Instanciadora("Prod");
                instancias[1] = new Instanciadora("Obj");
                for (int i = 0; i < 2; i++)
                System.out.print(instancias[i]);
                System.out.println();
        }
}
```

   Assinale a saída do programa:

   (a) Tipo: Prod, Id: 0; Tipo: Obj, Id: 1;

   (b) Tipo: Prod, Id: 1; Tipo: Prod, Id: 2;

   (c) Tipo: Prod, Id: 1; Tipo: Obj, Id: 2;

   (d) Tipo: Prod, Id: 2; Tipo: Obj, Id: 2;

   (e) Tipo: Obj, Id: 2; Tipo: Obj, Id: 2;

Matrícula:	Aluno:

2. Instâncias de classes são inicializadas por construtores. Assinale a afirmação correta sobre construtores de classes:

   (a) Uma classe pode ter mais de um construtor, somente se eles possuírem diferentes números de argumentos.

   (b) Uma classe pode ter mais de um construtor, mas apenas um deles pode ser do tipo **void**.

   (c) Uma classe pode ter mais de um construtor, desde que eles possuam um número diferente de argumentos, ou os argumentos possuam tipos diferentes.

   (d) Uma classe pode ter mais de um construtor, desde que todos possuam nomes diferentes.

   (e) Uma classe pode ter mais de um construtor, desde que eles sejam públicos e não estáticos.

3. Apesar do método destrutor **public void** finalize() poder ser utilizado para resolver pendências como um arquivo aberto ou uma conexão de banco de dados não finalizada, porque não devemos nos basear no destrutor para liberar recursos limitados de armazenamento ou de acesso a banco?

4. Utilizando testes unitários com JUnit, atualize a classe Data desenvolvida na Lista de Exercícios 1 para que ela possua métodos de teste associados à ela em todas as suas funcionalidades.

5. Em seguida, utilizando o TDD, atualize a classe Data desenvolvida na questão anterior para que seja possível escrever a data por extenso, a partir do método:

   ```
   public String porExtenso()
   ```

   Por exemplo, (**new** Data(15/11/1889)).porExtenso() deve retornar `15 de novembro de 1889`. Faça o uso adequado de atributos estáticos e finais. Escreva a nova classe Data na notação UML.

6. Crie uma classe de teste baseada na JUnit para a classe Conta desenvolvida na Lista de Exercícios 1 com casos de teste para os atributos desenvolvidos. Em seguida, utilizando a abordagem de desenvolvimento dirigido por teste, complemente a classe Conta desenvolvida anteriormente, adicionando aos atributos da classe um identificador contador, utilizando membros estáticos. Lembre-se de encapsular os dados. Desenvolva uma classe aplicação para ilustrar o funcionamento da classe Conta. Atualize a notação UML da classe Conta.

7. Transforme a classe Conversora em uma classe utilitária estática. Desenvolva casos de teste baseados na JUnit para os métodos da classe.