Ministério da Educação
Universidade Federal de Ouro Preto (Ufop)
Departamento de Computação e Sistemas (Decsi)

Professor: Rafael Alexandre
rfalexandre@ufop.edu.br
Disciplina: CSI032

Trabalho Prático 03 (TP03)

Instruções:

i - O arquivo deve ser entregue em formato .ZIP ou .RAR seguindo a nomenclatura: "XXXX.KKK" onde XXXX é o número de sua matrículo e KKK a extensão do arquivo.

ii - Cada um dos exercícios deve criado em um diretório com o seguinte nome: Exercicio_XX onde XX é o número da questão solucionada.

iii - Para cada programa desenvolvido deverão ser entregues **SOMENTE** os arquivos de projeto e classes Java em seus respectivos pacotes.

iv - O arquivo deve ser enviado via moodle limitado a data e hora de entrega definida no Plano de Ensino. Não serão aceitos trabalhos enviados por e-mail.

Questão 1. Considerando a referência **DEITEL P. J., DEITEL H. M., Java: como programar, 8a edição, São Paulo: Prentice Hall, 2010**, resolva:

- Página 359: exercício [11.15 - 11.22].

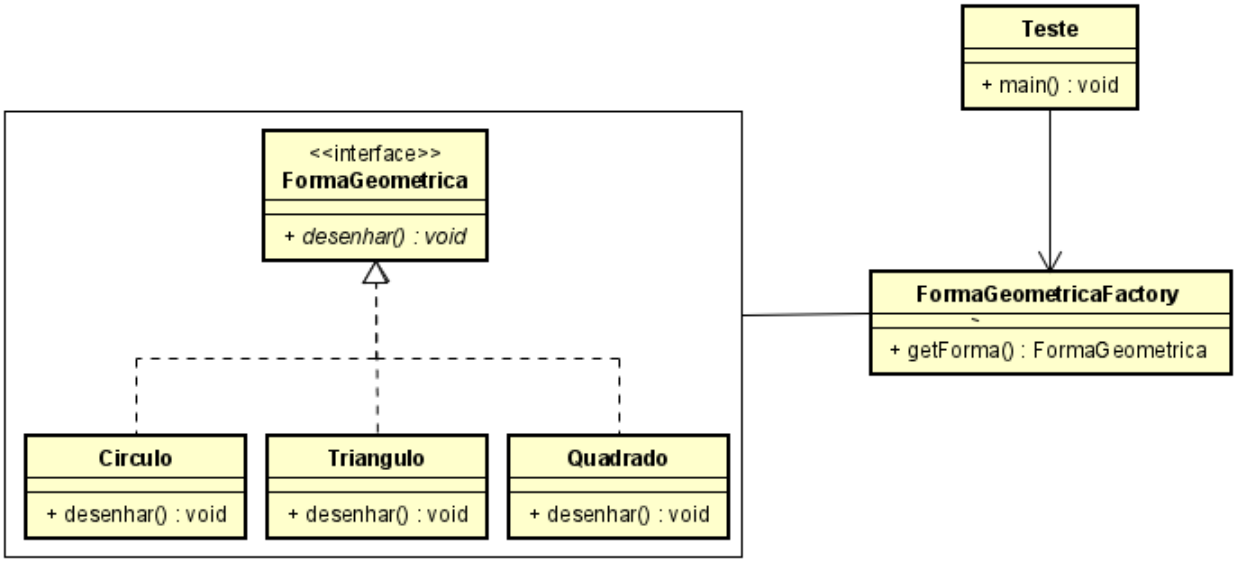
Questão 2. Algumas classes devem ser instanciadas uma única vez, crie um programa em JAVA no qual uma única instância de uma classe possa ser criada para toda aplicação.

Questão 3. Escreva, compile e execute o programa abaixo. Em seguida, troque sua implementação para que a classe Incremental seja Singleton. Execute novamente e veja os resultados.

```
class Incremental {
      private static int count = 0;
      private int numero;
      public Incremental() {
            numero = ++count;
      }
      public String toString() {
      return "Incremental " + numero;
      }
}
```

```
public class TesteIncremental {
      public static void main(String[] args) {
            for (int i = 0; i < 10; i++) {
                  Incremental inc = new Incremental();
                  System.out.println(inc);
            }
      }
}
```

Questão 4. Utilize o padrão Factory Method para a criação de formas geométricas. O método desenhar deve imprimir na tela o nome da forma geométrica.



Questão 5. Queremos montar uma lanchonete, onde vários sanduíches diferentes podem ser feitos. Um sanduiche básico contém: duas fatias de pão, uma fatia de queijo, uma fatia de presunto e salada (estrutura base de qualquer sanduiche). No entanto, existem variações do sanduiche básico de acordo com os tipos diferentes de ingredientes (veja quadro). Tendo o código para o sanduíche base, aplique o padrão de projeto Factory Method para que os sanduíches abaixo possam ser feitos na nossa lanchonete.

		Sanduiches		
Ingredientes		**Lanchonete CG**	**Lanchonete JP**	**Lanchonete RT**
Pão	**Integral**	X		
	Francês		X	
	Bola			X
Queijo	**Prato**	X		
	Mussarela		X	
	Cheddar			X
Presunto	**De Frango**	X	X	
	De Peru			X
Salada	**Com verdura**		X	
	Sem verdura	X		X

Questão 6. Utilize o padrão Abstract Factory para a criação de uma fábrica de carros. O sistema pode construir carros de dois tipos: Sedan ou Popular.

A) Sedan

i - Siena – Fiat;

ii - Fiesta Sedan – Ford;

A) Popular

i - Palio – Fiat;

ii - Fiesta – Ford;

Dica: Utilize o diagrama de classes 1 como referência.

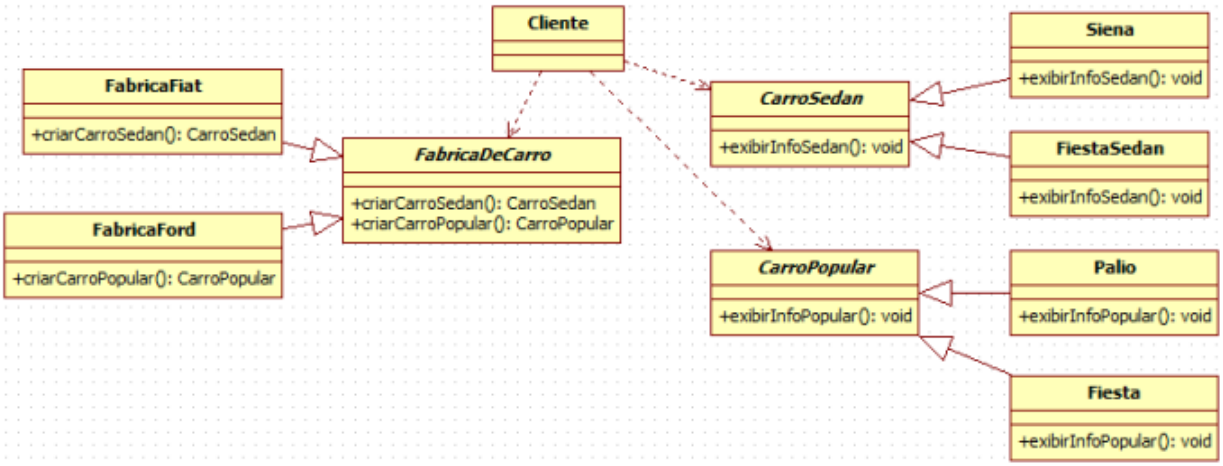


Figura 1: Fabricas de carros

Questão 7. Refine o código elaborado para a questão 4 com o objetivo de aplicar o padrão de projeto Abstract Factory. Para tanto, faz-se necessário:

A) Definir uma interface SanduichesIngredientFactory com os respectivos métodos de criação (Factory Methods);

B) Defina uma interface para cada tipo de produto: PãoIF, por exemplo;

C) Criar as fábricas concretas que devem implementar a interface definida:

i - SanduichesIngredientFactoryCG, SanduichesIngredientFactoryJP e FabricaDeSanduichesRT;

ii - Os Factory Methods devem ser implementados nas respectivas fábricas;

D) Modifique o método para montagem do sanduiche, para que o mesmo passe a receber como parâmetro a fábrica especifica;

E) Com as alterações sugeridas a Lanchonete será a responsável por criar o sanduiche e terá, por composição, uma fábrica de sanduiches;

F) Modifique, ainda, a classe principal, main, para que as modificações sejam refletidas.

Questão 8. Abaixo estão os códigos fonte de um cliente, uma interface para um somador que ele espera utilizar e uma classe concreta que implementa uma soma, mas não da maneira esperada pelo cliente. Como você pode ver abaixo, o cliente espera usar uma classe que soma inteiros em um vetor, mas a classe pronta soma inteiros em uma lista. Crie um adaptador para resolver esta situação.

```
public class Cliente {
	private SomadorEsperado somador;
	private Cliente(SomadorEsperado somador) {
		this.somador = somador;
	}
	public void executar() {
		int[] vetor = new int[] {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
		int soma = somador.somaVetor(vetor);
		System.out.println("Resultado: " + soma);
	}
}
```

```
public interface SomadorEsperado {
	int somaVetor(int[] vetor);
}
```

```
import java.util.List;
public class SomadorExistente {
	public int somaLista(List<Integer> lista) {
		int resultado = 0;
		for (int i : lista) resultado += i;
			return resultado;
	}
}
```

Questão 9. Suponha que você está implementando o sistema de uma estação meteorológica brasileira cuja medida utilizada para temperatura é *Celsius*. Uma das interfaces do seu sistema é a interface *MedidorCelsiusIF*, cujo método *medirTemperatura()* retorna sempre um valor double referente à temperatura na **medida Celsius**. O gerente do projeto decidiu comprar o termômetro no exterior, e, portanto este equipamento só mede no **formato Farenheit**. Abaixo veja a classe *MedidorFarenheit* da lib adquirida e o diagrama de classes do projeto.

A) a classe *MedidorFarenheit* da lib adquirida já está compilada – não podemos alterá-la.

B) $C = (F - 32)/1.8$: fórmula para converter a temperatura de Farenheit para Celsius

C) Crie um Adaptador para utilizar a biblioteca comprada.

```
import java.util.Random;
public class MedidorFarenheit {
      public double getTemperaturaFarenheit(){
            return new Random().nextDouble(); //simulando o termometro
      }
}
```