Lab V.

Objetivos

Os objetivos deste trabalho são:

- Identificar e utilizar padrões relacionados com a construção de objetos
- Aplicar boas práticas de programação por padrões em casos práticos

V.1 Serviço de comidas Take Away

Pretende-se criar um pequeno programa que dado um conjunto conhecido de comidas escolha o recipiente mais adequado ao seu transporte. Considere que os diferentes tipos de comidas são definidos segundo a seguinte interface.

```
public interface Commodity {
        public Temperature getTemperature();
        public State getState();
}
public enum State {
        Solid, Liquid;
}
public enum Temperature {
        WARM, COLD;
}
```

Deve criar um conjunto de classes que modele as seguintes comidas:

Classe	Estado	Temperatura	Outros
Milk	Liquid	Warm	
FruitJuice	Liquid	Cold	FruitName
Tuna	Solid	Cold	
Pork	Solid	Warm	

Analogamente, a descrição dos recipientes deve seguir a seguinte interface. Devem também ser criadas as classes para modelar os seguintes recipientes.

```
public abstract class Container {
        protected Commodity commodity;

        public boolean placeCommodity(Commodity c){
                this.commodity = c;
                return true;
        }
}
```

Classe	Adequado a:	
	Estado	Temperatura
PlasticBottle	Liquid	Cold
TermicBottle	Liquid	Warm, Cold
Tupperware	Solid	Warm, Cold
PlasticBag	Solid	Cold

Modele as classes e construa o código necessário para que o cliente possa executar

pedidos como os apresentados no método *main* seguinte. Deverá criar os dois métodos fábrica tendo em conta os requisitos de temperatura de estado do alimento de maneira a condicionar os alimentos de forma perfeita.

```
public static void main(String[] args) {
        Commodity[] menu = new Commodity[2];
        menu[0] = BeverageFactory.createBeverage(Temperature.COLD);
        menu[1] = MeatFactory.createMeat(Temperature.WARM);

        Container[] containers = new Container[2];
        containers[0] = ContainerFactory.createContainerFor(menu[0]);
        containers[1] = ContainerFactory.createContainerFor(menu[1]);

        containers[0].placeCommodity(menu[0]);
        containers[1].placeCommodity(menu[1]);

        System.out.println("Thank you for choosing your meal!");
        for(Container c : containers){
                System.out.println(c);
        }
}
```

Output:

```
Thank you for choosing your meal!
PlasticBottle [commodity=FruitJuice [fruit=Orange Temperature()=COLD, State()=Liquid]]
Tupperware [commodity=Pork [Temperatura()=WARM, State()=Solid]]
```

V.2 Fornecedor de almoços no campus universitário

Pretende-se criar um conjunto de classes que modele a elaboração de ementas no campus universitário. Para tal, considere que um almoço é representado pela classe *Lunch*.

```
class Lunch {
    private String drink;
    private String mainCourse;
    private String side;

       //.. restantes métodos
}
```

Considere ainda que todos os almoços são construídos seguindo um padrão *Builder* que usa a interface *LunchBuilder*.

```
interface LunchBuilder {
    public void buildDrink();
    public void buildMainCourse();
    public void buildSide();
    public Lunch getMeal();
}
```

Modele as classes e construa o código necessário para que o cliente possa executar pedidos como os apresentados no método *main* seguinte. Note que o código necessário para construir cada almoço é sempre o mesmo, apenas variando o *LunchBuilder* passado em *LunchDirector*.

```
public static void main(String[] args) {
    LunchBuilder lunch = new CrastoLunchBuilder();
```

```
        LunchDirector mealDirector = new LunchDirector(lunch);
        mealDirector.constructMeal();
        Lunch meal = mealDirector.getMeal();
        System.out.println("Ana's meal is: " + meal);

        mealDirector = new LunchDirector(new SnackLunchBuilder());
        mealDirector.constructMeal();
        meal = mealDirector.getMeal();
        System.out.println("Rui's meal is: " + meal);

        mealDirector = new LunchDirector(new CentralCantineLunchBuilder());
        mealDirector.constructMeal();
        meal = mealDirector.getMeal();
        System.out.println("My meal is: " + meal);
    }
```

Output:

```
Ana's meal is: [ drink: Vinho Tinto, main course: Bacalhau à lagareiro, side: Broa ]
Rui's meal is: [ drink: Sumo, main course: Pão com Panado, side: Rissol ]
My meal is: [ drink: Água, main course: Grelhada mista, side: Queijo fresco ]
```

V.3 Construtor com demasiados parâmetros

Considere a classe seguinte. Reescreva-a usando o padrão *builder*.

```
public class Movie {
   private final String title;
   private final int year;
   private final Person director;
   private final Person writer;
   private final String series;
   private final List<Person> cast;
   private final List<Place> locations;
   private final List<String> languages;
   private final List<String> genres;
   private final boolean isTelevision;
   private final boolean isNetflix;
   private final boolean isIndependent;

   public Movie(
      final String movieTitle,
      final int movieYear,
      final String movieDirector,
      final String movieWriter,
      final String movieSeries,
      final List<Person> movieCast,
      final List<Place> movieLocations,
      final List<String> movieLanguages,
      final List<String> movieGenres,
      final boolean television,
      final boolean netflix,
      final boolean independent)    {
             this.title = movieTitle;
             this.year = movieYear;
             this.director = movieDirector;
             this.writer = movieWriter;
             this.series = movieSeries;
             this.cast = movieCast;
             this.locations = movieLocations;
             this.languages = movieLanguages;
             this.genres = movieGenres;
             this.isTelevision = television;
```

```
                this.isNetflix = netflix;
                this.isIndependent = independent;
        }
    }
```

V.4 Classe Calendar

Analise a implementação da classe *java.util.Calendar* e identifique padrões de construção usados nesta classe. *Nota*: pode consultar este código em
http://www.docjar.com/html/api/java/util/Calendar.java.html
ou em
http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/Calendar.java
Reporte as suas observações no ficheiro *lab05/calendar.txt*.