ESQUEMA AULA PRÁTICA 10

❑ Herança e Classes Abstractas

Uma classe é ***abstracta*** se pelo menos um dos seus métodos de instância não se encontra implementado (só declarado sintacticamente). A primeira implicação é que uma tal classe não poderá ser instanciada... Porém o mecanismo de herança ainda vigora. Sendo assim qual a utilidade?

- Normalização e consequente redução de vocabulário: todas as subclasses concretas de uma classe abstracta terão que implementar os métodos que foram declarados como abstractos na classe abstracta.
- Todas as futuras implementações de subclasses de uma classe abstracta utilizarão uma linguagem comum. O polimorfismo garantirá que a resposta dada por uma qualquer instância de uma subclasse à mensagem recebida é a apropriada.

1. Defina uma classe abstracta, `Count`, para representar contadores, com operações para incrementar de uma unidade, decrementar de uma unidade, inspeccionar o valor corrente e reinicializar a zero. Defina classes derivadas `CountPositive` em que a operação para decrementar não tem efeito se o contador estiver a zero e `CountModulus` que volta a zero quando atinge um valor máximo predeterminado, e volta a esse valor máximo menos um ao decrementar zero.

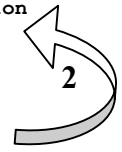
2. Estude as API da classe abstracta `Number` e das suas subclasses `Double`, `Float`, `Integer` e `Long`. Faça um programa para testar os métodos disponibilizados.

NOTA: Deve ter atenção aos métodos que lançam (*throws*) excepções (sinal gerado pela JVM em tempo de execução indicando uma situação de erro da qual é possível recuperar – e.g. *NumberFormatException*). Uma vez que estes não fazem o tratamento da excepção cabe ao método chamador tratá-la (através da cláusula ***catch***) ou comunicar (através da cláusula ***throws***) ao seguinte método da pilha de execução que poderá lançar a dita excepção. Atente-se no seguinte exemplo:

```
import java.io.IOException;

public class Principal {

  public static void main(String[] args) throws IOException, NumberFormatException
  {
       System.out.println (le_inteiro());
  }
```



```
  static int le_inteiro () throws IOException, NumberFormatException {
        String s=" ";
        char c;

        System.out.print("Valor inteiro= ");
        while ( (c=(char)System.in.read())!= 10)    //lança IOException
                s+=c;
        s=s.trim();
        return (Integer.valueOf(s).intValue());//para as RunTimeExceptions
                                        //onde se inclui a NumberFormatException
                                        //pode omitir-se a comunicação
  }
}
```

3. Programe um método para trocar dois valores inteiros entre si. Terá que "embrulhar" cada um deles num objecto que permita alterar (*setValue*) o valor inteiro armazenado. A classe `Integer` não serve, pois não? Caso a resposta anterior seja negativa programe a classe `MyInteger` de forma a poder responder ao que é pedido. A classe `MyInteger` será subclasse de que classe?

4. Apresente a sua solução para o seguinte problema: "É necessário um método para trocar dois números (de tipo **float**, ou **double**, ou **long**, ou **int**) entre si.". Programe a sua solução e teste-a para pares de valores numéricos com o mesmo tipo. Como funciona a sua solução para pares de tipo heterogéneo?

5. Estude a API da classe abstracta `AbstractCollection` presente no *package* `java.util`. Apresente em esquema a hierarquia de classes que dela resulta. Faça um programa para testar os métodos disponibilizados em uma das concretizações da referida classe.

6. Pense no conjunto de mensagens a que uma qualquer figura bidimensional (e.g. ponto, linha, círculo, elipse, triângulo, uma qualquer área poligonal fechada, etc.) deverá ser capaz de responder. Defina a classe (abstracta?) `Figura2D` e programe as subclasses sugeridas e outras que considere pertinentes.