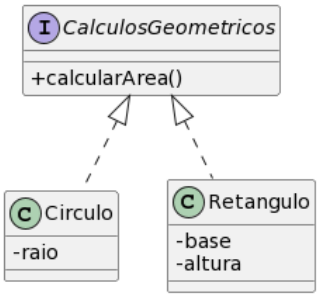
Ficha de trabalho #8

Herança Múltipla
Interfaces

O Java não permite **herança múltipla** (possibilidade de uma classe herdar características e comportamento de mais do que uma classe), mas é possível usar Interfaces para simular esse comportamento.
Uma **interface** é um tipo de dados sem implementação associada, declarando funcionalidades que serão implementadas nas classes que a utilizam. Assim, apenas pode conter constantes (identificadas pelas palavras reservadas static final) e métodos abstratos (definindo-se apenas a assinatura do método). Usa-se as interfaces para obrigar as classes que as utilizam a ter determinados comportamentos.
Num projeto semelhante ao da ficha#7, em que pretendemos representar várias figuras geométricas, cada uma com características diferentes (ex: raio do Circulo, base e altura do Retangulo), sem atributos comuns, mas queremos que todas tenham um método para calcular a área. Podemos assim ter uma interface CalculosGeometricos que define um método abstract calcularArea() que depois será implementado com regras diferentes em cada uma das classes que utilize a interface. Em baixo do lado esquerdo encontra-se o diagrama de classes do nosso projeto. Do lado direito encontra-se o código da interface.




```java
public interface CalculosGeometricos {
    public double calcularArea();
}
```

Depois de feita a interface, em cada uma das classes que a devem utilizar, iremos indicar que a classe implementa as funcionalidades definidas pela interface CalculosGeometricos (implements CalculosGeometricos), o que nos irá obrigar a implementar o método para calcular a área, com as regras específicas de cada tipo de figura geométrica:

```java
public class Circulo implements CalculosGeometricos {
    private double raio;

    // ...

    @Override
    public double calcularArea() {
        return Math.PI * raio * raio;
    }
}
```

```java
public class Retangulo implements CalculosGeometricos {
    private double base;
    private double altura;

    // ...

    @Override
    public double calcularArea() {
        return this.base * this.altura;
    }
}
```

No futuro, caso seja suposto que todas as figuras tenham mais algum método de cálculo geométrico (ex: calcularPerimetro()), basta-nos adicionar a assinatura do método na interface e o IDE irá lembrar-nos de todas as classes em que devemos implementá-lo.

Como as classes implementam a mesma interface, podemos criar um ArrayList de objetos do tipo da interface (da mesma forma que com a herança na ficha#7 podíamos criar um ArrayList de objetos do tipo Figura) e até usar o istanceOf para verificar se os objetos são de um determinado tipo.

```java
public class TesteFiguras {
    public static void main(String[] args) {
        Circulo c1 = new Circulo(1);
        Circulo c2 = new Circulo(2.5);
        Circulo c3 = new Circulo(10);
        Retangulo r1 = new Retangulo(1,1);
        Retangulo r2 = new Retangulo(2,5);
        Retangulo r3 = new Retangulo(10,3);

        ArrayList<CalculosGeometricos> figuras = new ArrayList<>();
        figuras.add(c1); figuras.add(c2); figuras.add(c3);
        figuras.add(r1); figuras.add(r2); figuras.add(r3);

        System.out.println("=== Figuras ===");
        for(CalculosGeometricos f: figuras) {
            System.out.println(f);
        }

        System.out.println("=== Circulos ===");
        for(CalculosGeometricos f: figuras) {
            if(f instanceof Circulo) {
                System.out.println(f);
            }
        }

        System.out.println("=== Retangulos ===");
        for(CalculosGeometricos f: figuras) {
            if(f instanceof Retangulo) {
                System.out.println(f);
            }
        }
    }
}
```

1. Pretende-se criar um programa em Java, seguindo as regras da POO e com documentação recorrendo a JavaDoc, para representar uma **escola** que é composta por professores e alunos, alguns dos quais são bolseiros, permitindo calcular o valor a pagar tanto aos professores como aos alunos bolseiros. Para cada uma das **pessoas** associadas à escola precisamos de saber o nome e o NIF. Para além disso, sobre os **professores** precisamos de guardar o salário base, valor por hora extra (igual para todos os professores, com valor inicial de 10€) e número de horas extra realizadas. Para os **alunos**, precisamos ainda de saber o curso e o ano em que estão inscritos e, se o aluno for um **aluno bolseiro**, precisamos ainda de saber o valor mensal da bolsa. O programa deve permitir:
   - Criar professores e alunos (alguns bolseiros), e adicioná-los a uma lista
   - Mostrar todas as pessoas da lista, incluindo o valor a receber, se aplicável
   - Mostrar todas as pessoas agrupadas por tipo

2. Pretende-se criar um programa em Java, seguindo as regras da POO e com documentação recorrendo a JavaDoc, para representar atrações turísticas, e calcular o preço da visita a algumas das atrações, que depende da idade do visitante e com diferentes regras de cálculo dependendo do tipo da atração. Todas as **atrações** têm um nome e um local, e podem ser museus ou parques (alguns são parques de diversões). Para os **museus** precisamos de guardar o preço do bilhete, a idade limite para obter desconto e a percentagem de desconto pela idade (estes valores são iguais para todos os museus, sendo que atualmente há um desconto de 10% para pessoas com mais de 65 anos). O preço da visita é o preço do bilhete, com a possibilidade de obter desconto caso a idade seja superior ao limite definido. Os **parques** são geralmente de entrada livre e a única informação extra necessária é a sua área (em m^2). Os **parques de diversões**, um subtipo de parque, têm informação sobre o número de diversões que contêm, o preço do bilhete (por diversão) sem risco e o preço do bilhete (por diversão) com risco, bem como a idade limite para que seja considerado que a pessoa está numa faixa etária de risco. Estes últimos 3 valores são iguais para todos os parques de diversões e atualmente uma pessoa com idade inferior a 18 anos paga 0.75€ por diversão, e uma pessoa com pelo menos 18 anos paga 0.55€. O preço da visita será o produto do número de diversões pelo preço por diversão, tendo em conta a idade do visitante. O programa deve permitir:
   - Criar uma lista de atrações de diferentes tipos
   - Mostrar todas as atrações da lista, juntamente com o preço da visita, se aplicável
   - Mostrar as atrações agrupadas pelo tipo

3. Pretende-se criar um programa em Java, seguindo as regras da POO e com documentação recorrendo a JavaDoc, para representar veículos, permitindo calcular o custo por quilómetro dos veículos motorizados. Todos os **veículos** têm uma marca e um modelo e podem ser de motorizados ou não motorizados. Os **veículos motorizados** têm informação sobre a potência do motor, o consumo (em L/100km) e o preço por litro de combustível (igual para todos os veículos motorizados, atualmente com o valor de 1.69€). O custo por quilómetro é calculado com base no consumo e no preço por litro. Os **veículos não motorizados** têm informação sobre o peso do veículo e a carga máxima. Neste caso, não há necessidade de calculo do custo por quilómetro. O programa deve permitir:
   - Criar uma lista de veículos de diferentes tipos
   - Mostrar todos os veículos e os custos por quilómetro, se aplicável
   - Mostrar os veículos agrupados por tipo