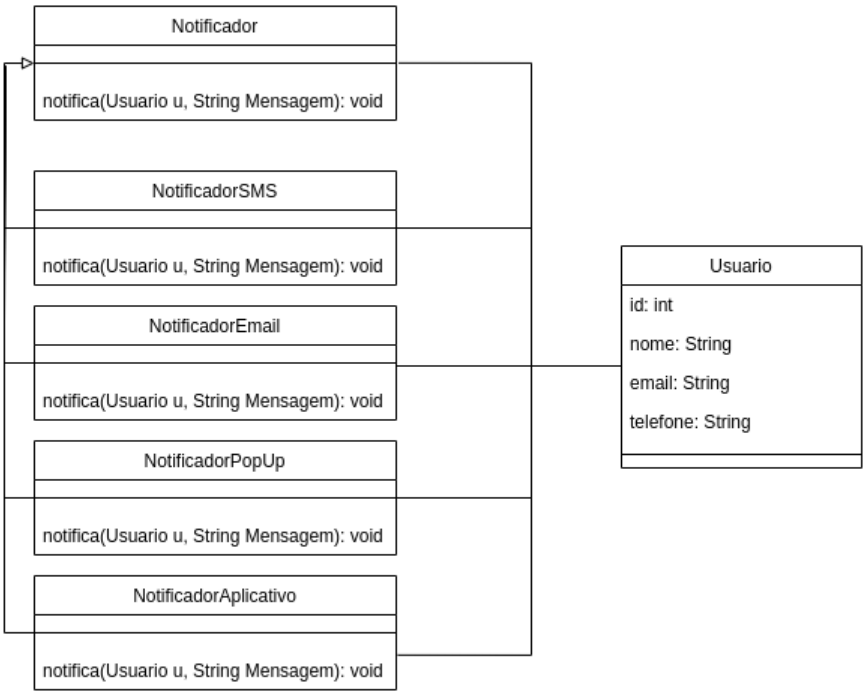
Aula de Laboratório – Classes Abstratas e Interfaces

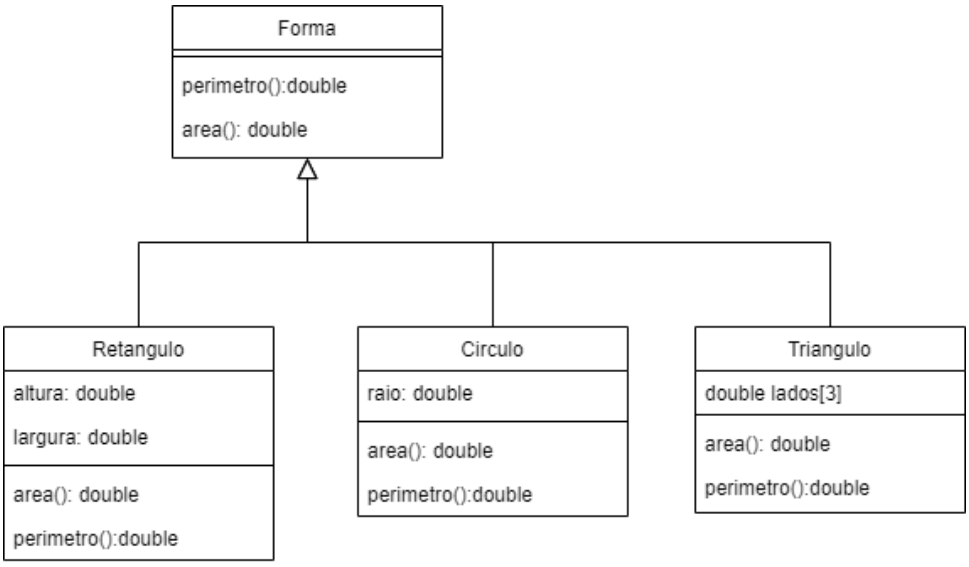
1) **Implementação de Interfaces**: Considere o problema de exibir os participantes de um concurso de forma ordenada pela nota.
    a. Crie uma classe Participante com os atributos nome, cpf e nota. Escreve o método toString e um construtor.
    b. Crie uma classe App com o método main. Nesta classe, crie um ArrayList para armazenar os participantes do concurso e adicione 3 participantes com notas diferentes.
    c. Para ordenar a lista usando as notas, pode ser usado o método sort existente na classe Collections. Para usá-la, primeiro importe no início do arquivo "java.util.Collections". Se o ArrayList se chamar "participantes", a ordenação pode ser feita usando "Collections.sort(participantes)". Contudo, se você tentar executar o programa, será exibido um erro dizendo que o ArrayList não pode ser usado como entrada para o sort. Isto acontece porque para ordenar a lista, é necessário dizer como os Participantes devem ser comparados, ou seja, qual critério deve ser usado para ordená-los. Esta restrição é adicionada definindo que apenas classes que implementam a interface Comparable podem ser usados como entrada para o sort.
    d. Atualize a classe Participante, diga que ela implementa a interface Comparable<Participante> e defina o método compareTo como indicado abaixo. Verifique que após esta mudança, o programa passa a funcionar.

```
public class Participante implements Comparable<Participante>
{
    String cpf;
    String nome;
    int nota;

    public Participante(String cpf, String nome, int nota) {
        this.cpf = cpf;
        this.nome = nome;
        this.nota = nota;
    }

    public String toString() {
        return "Pessoa(" + nome + ", cpf=" + cpf + ", nota=" +
nota + ")";
    }

    public int compareTo(Participante p) {
        return p.nota - nota;
    }
}
```

2) Implementação de Interfaces: Este exercício é uma continuação da atividade anterior. Crie uma classe Projeto que possui descrição (String), departamento (String), duração (int) e valor (double). Crie um ArrayList de projetos e ordene os projetos pelo valor de forma decrescente. Quando os custos forem iguais, deve vir primeiro o projeto com menor duração.

3) Criação de Interfaces: Ajuste os exemplos dos notificadores e das formas da aula passada de forma que Notificador e Forma sejam interfaces. Crie ArrayLists e verifique que é possível adicionar objetos das subclasses nos Arrays e invocar os métodos definidos na interface em um comando de repetição (no caso dos notificadores, invocar o método notifica para cada elemento do array em um for).





Observação: para calcular a área do triângulo a partir dos lados, use a fórmula abaixo:

Outras fórmulas para calcular a área de um triângulo

Existe outro método para calcular a área de triângulos conhecido como **fórmula de Heron**. Utilizamos essa fórmula quando conhecemos apenas a medida dos lados do triângulo, mas não conhecemos a altura. Para aplicar a fórmula de Heron do triângulo de lados a, b e c, **primeiro calculamos o semiperímetro**, ou seja, metade do perímetro do triângulo.

$$p = \frac{a+b+c}{2}$$

Conhecendo o valor do semiperímetro, basta utilizar a fórmula:

$$A = \sqrt{p(p-a)(p-b)(p-c)}$$

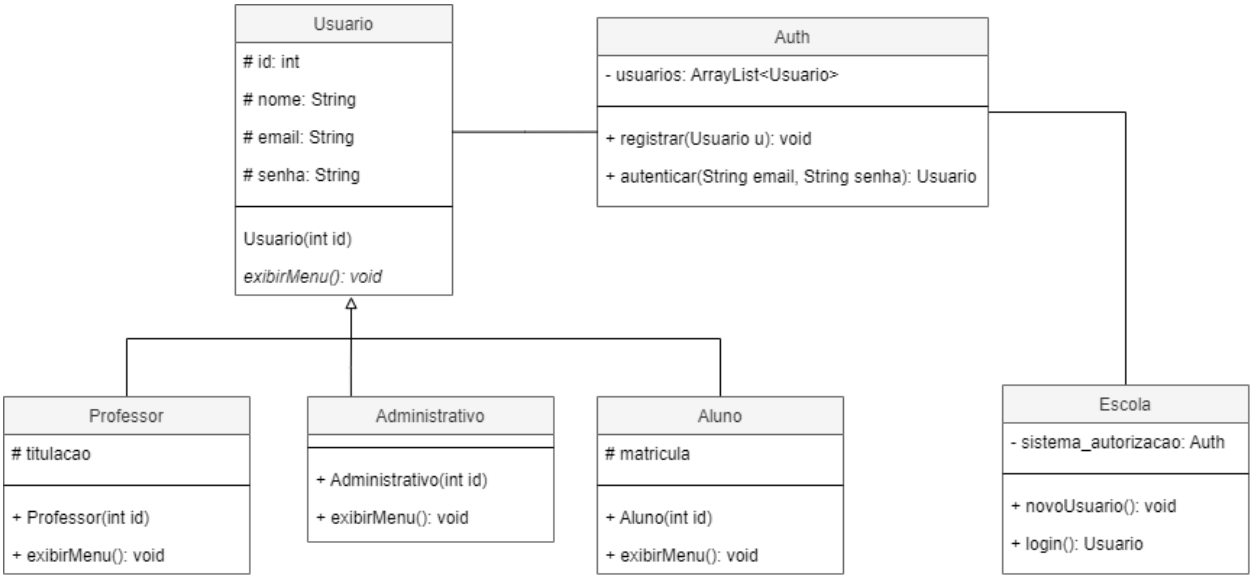
Figure 1. Extraída de https://mundoeducacao.uol.com.br/matematica/area-triangulo.htm

4) Classes Abstratas: Esta atividade é baseada no sistema de autorização visto nas aulas passadas. É comum que diferentes perfis de usuários sejam capazes de usar sistemas de diferentes formas. Em uma escola, por exemplo, considere que:

- Professores são capazes de registrar presenças e notas;
- Alunos podem se matricular em disciplinas, visualizar materiais, enviar mensagens para o professor e realizar exercícios;
- Técnicos administrativos podem registrar disciplinas e atribuir professores à disciplinas.

Neste exercício **NÃO** vamos implementar todas estas funcionalidades, mas criar um subsistema para cadastro de usuários e que permita exibir diferentes opções para diferentes usuários. Para isto, implemente a arquitetura de classes considerando as restrições abaixo:

- A classe Usuário contém um construtor público que deve ler os valores de todos os atributos exceto o identificador que deve ser passado como argumento. O método exibirMenu() é abstrato e será implementado nas subclasses.
- Os construtores das classes Professor, Administrativo e Aluno devem usar o construtor da classe Usuário para ler os dados básicos (nome, e-mail, etc.) e, quando necessário, ler os atributos específicos da classe (titulação no Professor e matrícula no Aluno). Os métodos exibirMenu() destas classes devem exibir um menu com as opções disponíveis às pessoas daquele perfil.
- O método registrar da classe Auth deve adicionar um usuário no ArrayList. O método autenticar deve retornar o usuário com o e-mail e senha passados, ou null caso não exista nenhum.
- O método novoUsuario da classe Escola deve solicitar que o usuário do programa escolha um tipo de perfil (Aluno, Professor ou Administrativo) para criar um novo cadastro, criar um objeto da classe escolhida e registrar o usuário no sistema de autorização.
- O método login da classe Escola deve solicitar um e-mail e senha, recuperar o respectivo usuário usando o método Auth.autenticar e exibir o menu de funcionalidades do usuário. Não é necessário solicitar que seja escolhida uma opção. É apenas necessário mostrar as possibilidades.
- O programa principal deve perguntar se o usuário do programa deseja registrar um novo usuário ou fazer login e invocar as funções apropriadas.


Usuario
# id: int
# nome: String
# email: String
# senha: String
Usuario(int id)
exibirMenu(): void
Auth
- usuarios: ArrayList<Usuario>
+ registrar(Usuario u): void
+ autenticar(String email, String senha): Usuario
Professor
titulacao
+ Professor(int id)
+ exibirMenu(): void
Administrativo
+ Administrativo(int id)
+ exibirMenu(): void
Aluno
matricula
+ Aluno(int id)
+ exibirMenu(): void
Escola
- sistema_autorizacao: Auth
+ novoUsuario(): void
+ login(): Usuario