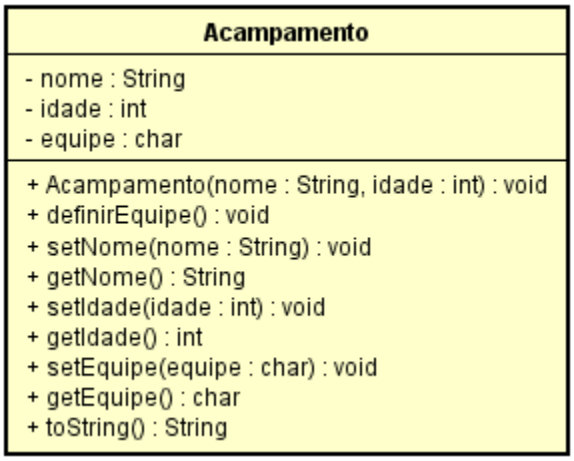
# Exercício 4

## ATIVIDADE 1

Desenvolver a seguinte classe:




A seguir, uma breve descrição dos métodos é apresentada:

| **Método** | **Descrição** |
|---|---|
| `Acampamento` | Construtor com 2 parâmetros: `nome` e `idade`. Deverá chamar os métodos setNome e setIdade. |
| `definirEquipe` | Este método não retorna valor e deverá verificar as seguintes condições: se a idade estiver entre 6 e 10 anos, a equipe será A; se a idade estiver entre 11 e 20, a equipe será B; se a idade for 21 ou mais, a equipe será C. Chamar o método setEquipe. |
| `setNomeDoAtributo` | Método modificador que recebe um parâmetro e o atribui ao respectivo atributo. |
| `getNomeDoAtributo` | Método de acesso que retorna o valor do respectivo atributo. |
| `toString` | Método que retorna uma string concatenando a exibição dos atributos nome, idade e equipe. |

Em seguida, criar uma classe de teste que contenha um vetor de 10 objetos da classe `Acampamento` (vetor estático). Para cada indivíduo, ler o nome e a idade e passar estas informações para o construtor da classe Acampamento, criando um novo objeto, armazenando no vetor e chamando o método `definirEquipe()`. Em seguida, imprimir um resumo contendo o nome, idade e a equipe de cada objeto.

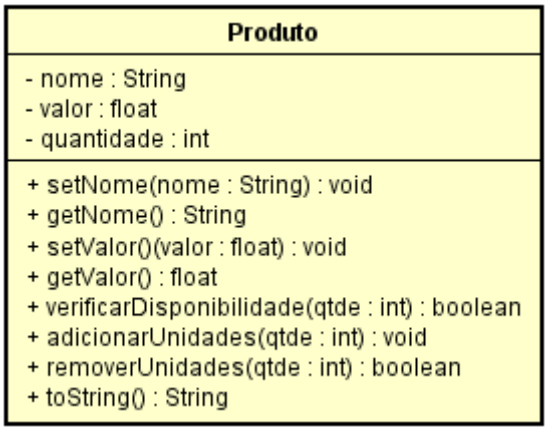
Dicas:

- como temos o método `toString` na classe Acampamento, basta dar um println na própria referência do objeto para obter a string retornada por `toString`;
- você pode definir o tamanho do vetor como um atributo da classe com valor constante, conforme exemplo abaixo. O tamanho do vetor também está disponível em seu atributo length.

```
final int x = 10;
```

## ATIVIDADE 2

Desenvolver a seguinte classe:




Os métodos são descritos brevemente abaixo:

| **Método** | **Descrição** |
|---|---|
| `setNomeDoAtributo` | Método modificador que recebe um parâmetro e o atribui ao respectivo atributo. |
| `getNomeDoAtributo` | Método de acesso que retorna o valor do respectivo atributo. |
| `verificarDisponibilidade` | Deve retornar um valor boolean e receber um parâmetro inteiro contendo a quantidade que se quer verificar. Retorna true caso exista a quantidade solicitada do produto no estoque, e false caso contrário. |
| `adicionarUnidades` | Não retorna valor e recebe como parâmetro a quantidade a ser acrescentada à quantidade atual. |
| `removerUnidades` | Retorna um boolean e deverá receber como parâmetro a quantidade de produtos a serem removidos. Antes da remoção, deve-se verificar se há disponibilidade do produto solicitado. Para isso, executar o método |

| | |
|---|---|
| | verificarDisponibilidade(), e caso este retorne true, o método removerUnidades() deverá diminuir o valor recebido como parâmetro da quantidade em estoque. Retornar true caso a remoção tenha sido executada com sucesso, e false caso não haja disponibilidade da quantidade a ser removida. |
| `toString` | Retorna uma string concatenando nome, valor e quantidade |

Faça um programa principal que crie um ArrayList de Produtos.

Deve ser exibido um menu com as opções *1 – Cadastrar Produto, 2 – Consultar estoque*, 3 – *Remover unidades*, 4 – *Adicionar unidades* e *9 - Sair*.

Para consultar um produto, remover ou adicionar unidades, deve-se solicitar primeiramente o código do produto (o índice do vetor). Por exemplo, para consultar a disponibilidade do produto 2, acessar o objeto armazenado na posição 2 do vetor.

**Dicas**:

- Para melhor organizar o código do programa principal, o código pode ser quebrado em vários métodos. No método main, deve-se instanciar um objeto da própria classe de teste e utilizá-lo para chamar os demais métodos.

Ex:

```
public class ClasseTeste {
	public void metodoA () {
		...
	}
	public void metodoB() {
		...
	}
	public static void main (String[] args) {
		ClasseTeste obj = new ClasseTeste();
		obj.metodoA();
		obj.metodoB();
	}
}
```

- Para ler uma String após ter lido um número com o scanner, é necessário dar um nextLine() para “limpar” o caractere de quebra de linha (enter), caso contrário, teremos problemas na leitura da String.