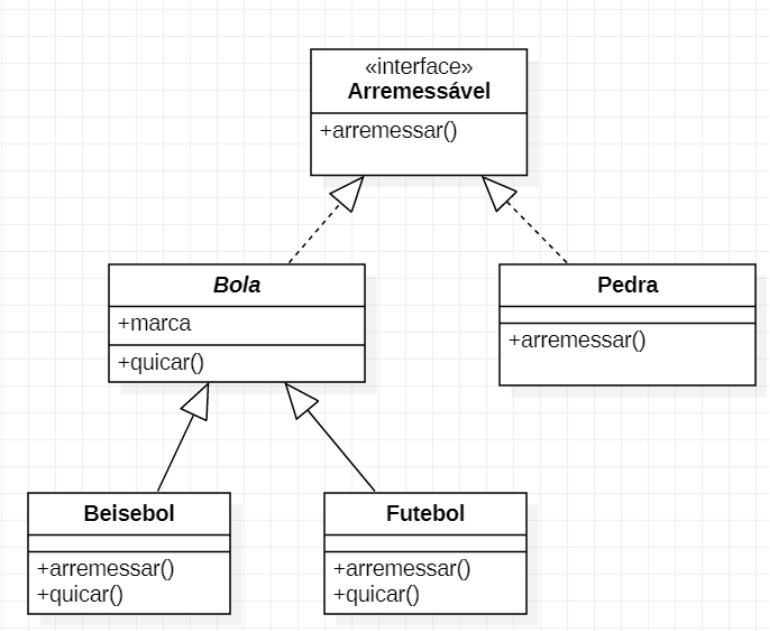
Lista de Exercícios Java:

Viviane Pinto de Souza (20191015030260)

Questão 1: Implemente a seguinte hierarquia de classes no editor. Você pode preencher os corpos do método para os métodos arremessar ou quicar com simples System.out.println.




```
package questaoum;
public interface Arremessavel {

	void arremessar();
}
```

```
package questaoum;
public abstract class Bola implements Arremessavel {

	public String marca;

	public void quicar() {

		System.out.println("QUICANDO!");
	}

}
```

```
package questaoum;
public class Pedra implements Arremessavel{

	@Override
	public void arremessar() {
		System.out.println("AREEMENSSANDO!");
	}
}
```

```
package questaoum;
public class Beisebol extends Bola{
	@Override
```

```
	public void arremessar() {
		// TODO Auto-generated method stub
		System.out.println("AREEMENSSANDO!");

	}

	public void quicar() {

		System.out.println("QUICANDO!");

	}
}
```

```
package questaoum;
public class Futebol extends Bola{
	@Override
	public void arremessar() {
		// TODO Auto-generated method stub

		System.out.println("AREEMENSSANDO!");

	}

	public void quicar() {

		System.out.println("QUICANDO!");

	}
}
```

```
package questaoum;
public class Main {
   public static void main(String[] args) {
	// TODO Auto-generated method stub
	Beisebol bola1 = new Beisebol();
	bola1.arremessar();
	bola1.marca = "Beisebolas";
	bola1.quicar();
	// Acesso à variável marca usando a referência à instância (bola1)
	System.out.println("A bola de beisebol é da marca " + bola1.marca + "." );
   }
}
```

Responda as seguintes questões:

·Em que classes abstratas e interfaces são iguais?

resposta: Amabas podem ter métodos abstratos e concretos, onde o primeiro é sem implementação e o segundo com implementação, além disso também podem ter membros e fornecer um mecanismo de herança.

·Em que classes abstratas e interfaces são diferentes?

resposta: Classes abstratas podem ter construtores, enquanto interfaces não. Os membros (campos e métodos) de uma interface são implicitamente públicos, estáticos e finais, já nas classes abstratas a visibilidade dos membros pode ser definida conforme necessário, e esses membros podem ser estáticos, finais ou abstratos. Ainda, nas classes abstratas os métodos podem ser mistos (abstratos e/ou concentros), enquanto na interface, todos os métodos são implicitamente abstratos, mas a partir do Java 8, você pode ter métodos default (com implementação) e métodos estáticos. A palavra chave da classe abstrata é “abstract”, enquanto da interface é “interface. Uma classe abstrata é estendida por meio da palavra chave “extends”, quando a interface, “implements”.

·Preencha a tabela a seguir com um dos três valores.

· ✓ Um objeto do tipo indicado pode ser armazenado em uma variável do tipo indicado.

· **X** Um objeto do tipo indicado não pôde ser armazenado em uma variável do tipo indicado.

·— Não é possível instanciar um objeto do tipo indicado.

TIPO DE OBJETO	Tipo de Variável	Tipo de Variável	Tipo de Variável	Tipo de Variável	Tipo de Variável
	Arremessável	Bola	Pedra	Beisebol	Futebol
Arremessável	—	—	—	—	—
Bola	—	—	—	—	—
Pedra	✓	X	✓	X	X
Beisebol	✓	✓	X	✓	X
Futebol	✓	✓	X	X	✓

.

.

.

Qual dos seguintes irá **compilar** (marcar com um **C**)? Qual dos seguintes será executado com sucesso (marque com um **E**)?

1. Bola Bola = new Futebol();

   C E

2. Bola Bola = new Futebol();

   Beisebol beisebol = (Beisebol)bola;

NEM UM, NEM OUTRO! Java diferencia LETRAS maiúsculas de minúsculas, por isso não reconhece a variável bola a ser convertida.

3. Object obj = new Beisebol();

   C E

4. Object obj = new Beisebol();

   Arremessavel arremessavel = obj;

   NEM UM, NEM OUTRO! Para ser colocado em uma variável mais especializada, é preciso fazer uma conversão de tipo explícita.

5. Arremessavel arremessavel = new Beisebol(); Objeto obj = arremessavel;

C E

Questão 2: Faça o que se pede em relação aos seguintes assuntos.

a. Herança Múltipla:

- Explique por que o Java não suporta herança múltipla de classes, e como você pode contornar essa limitação usando interfaces.

Por simplicidade, clareza e para evitar problemas reclaionados ao “Problema Diamante” que é quando uma classe herda de duas classes que, por sua vez, herdam de uma mesma classe pai. Isso gera uma ambiguidade quando a classe derivada tenta herdar métodos ou atributos da classe pai compartilhada.

b. Interfaces e Classes Abstratas:

- Considere uma hierarquia de classes com uma classe abstrata `Veiculo` e interfaces `Eletrico` e `Conduzivel`. Como você projetaria uma classe concreta `CarroEletrico` que implementa ambas as interfaces?

herdando as características gerais de Veículo, mas também implementando funcionalidades únicas das Interfaces.

c. Polimorfismo:

- Crie uma interface `Imprimivel` com um método `imprimirDetalhes()`. Implemente essa interface em duas classes, `Livro` e `Pessoa`. Em seguida, crie um método que aceita objetos de qualquer classe que implementa `Imprimivel` e chame o método `imprimirDetalhes()` para cada objeto.

```
package questaoum;
public interface Imprimivel {


    void imprimirDetalhes();
}
```

```
package questaoum;
public class Livro implements Imprimivel{

    private String titulo;
       private String autor;
       public Livro(String titulo, String autor) {
           this.titulo = titulo;
           this.autor = autor;
       }
       @Override
       public void imprimirDetalhes() {
           System.out.println("Livro: " + titulo + " - Autor: " + autor);
       }
}
```

```
package questaoum;
public class Pessoa implements Imprimivel{
```

```
    private String nome;
    private int idade;
    public Pessoa(String nome, int idade) {
        this.nome = nome;
        this.idade = idade;
    }
    @Override
    public void imprimirDetalhes() {
        System.out.println("Pessoa: " + nome + " - Idade: " + idade + " anos");
    }
}
```

```
package questaoum;
public class MainDois {
    public static void main(String[] args) {
            // Criando objetos Livro e Pessoa
            Livro livro = new Livro("O Hobbit", "J.R.R. Tolkien");
            Pessoa pessoa = new Pessoa("Viviane", 27);
            // Chamando o método imprimirDetalhes() para cada objeto
            livro.imprimirDetalhes();
            pessoa.imprimirDetalhes();
            // Criando um array de objetos Imprimivel
            Imprimivel[] objetos = {livro, pessoa};
            // Chamando o método que aceita objetos Imprimivel
            imprimirDetalhesImprimiveis(objetos);
        }
        // Método que aceita objetos Imprimivel e chama imprimirDetalhes()
        static void imprimirDetalhesImprimiveis(Imprimivel[] objetos) {
            for (Imprimivel objeto : objetos) {
                objeto.imprimirDetalhes();
            }
        }
}
```

d. Classe Abstrata e Construtores:

- Por que uma classe abstrata pode ter construtores? Explique e forneça um exemplo prático de uma situação em que um construtor em uma classe abstrata seria útil.

Os construtores em classes abstratas são úteis para garantir que certos atributos sejam configurados corretamente e que a inicialização comum seja realizada antes que as subclasses possam realizar suas próprias inicializações específicas. Um exemplo prático seria pegar o exemplo do item b. A classe abstrata Veículo representa TODOS os veículos, daí a gente pode ter um construtor nela que inicializa atributos comuns a CARRO e MOTO, por exemplo. Então nessas subclasses de veículos (que irão estender a classe abstrata Veículo), podem ser inicializados seus próprios atributos específicos.

e. Polimorfismo Dinâmico (ou Late Binding):

- Explique o conceito de polimorfismo dinâmico em Java. Como ele difere do polimorfismo estático? Dê um exemplo prático mostrando como o polimorfismo dinâmico é alcançado em Java.

O polimorfismo dinâmico, também conhecido como ligação tardia ou polimorfismo de execução, ocorre em tempo de execução. Nesse tipo de polimorfismo, o método a ser executado é decidido no tempo de execução com base no tipo real do objeto. Ele se difere do Estático, pois este ocorre em tempo de compilação, refere-se à capacidade de uma classe ter métodos com o mesmo nome, mas com assinaturas diferentes. O estático é interessante para o Java saber qual método chamar durante a compilação.