Formabase
Aprenda passo a passo

Java
Interfaces

Metodologia

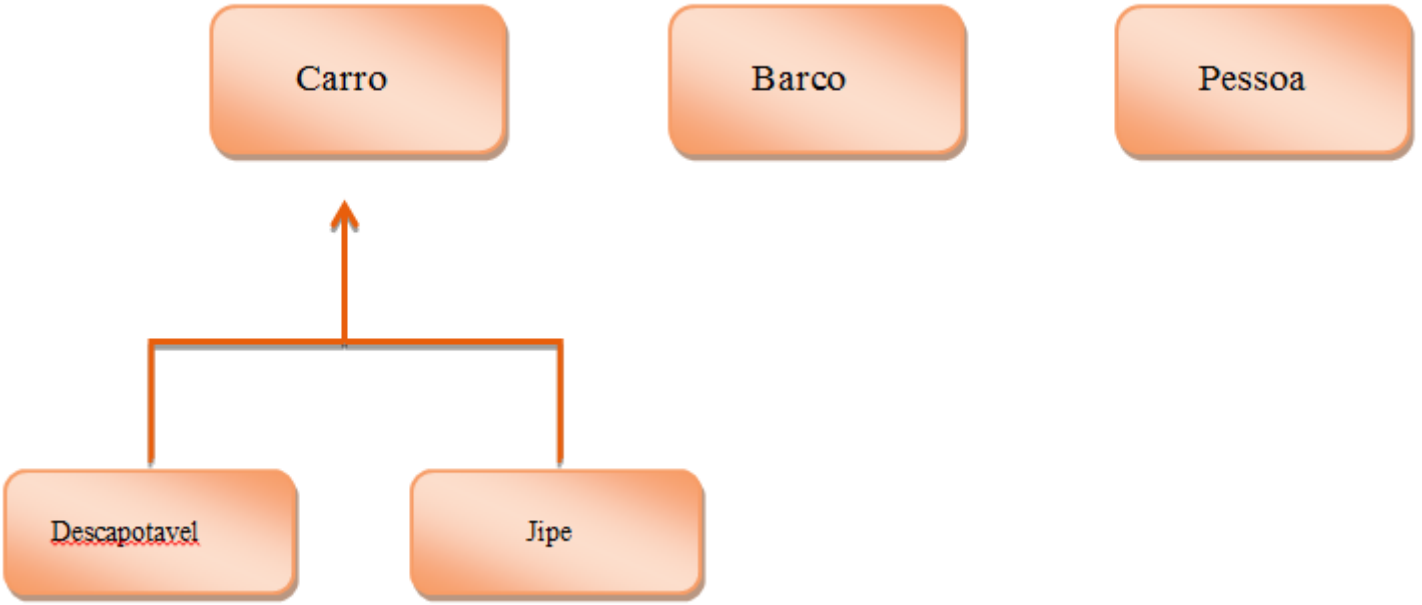
- Interprete o documento calmamente e com atenção.
- Acompanhe a execução do exercício no seu computador.
- Não hesite em consultar o formador para o esclarecimento de qualquer questão.
- Não prossiga para o ponto seguinte sem ter compreendido totalmente o ponto anterior.
- Caso seja necessário, execute várias vezes o exercício até ter compreendido totalmente o processo.

Conteúdo programático

1. Interfaces

Uma **interface** é uma entidade que define uma ou várias ações, ou seja, força uma estrutura que as classes têm que implementar. Diz-se que determinada classe implementa uma interface se essa classe definir todos os métodos que a interface especifica. As interfaces são úteis quando se pretende especificar ações iguais para entidades que não estão hierarquicamente relacionadas.

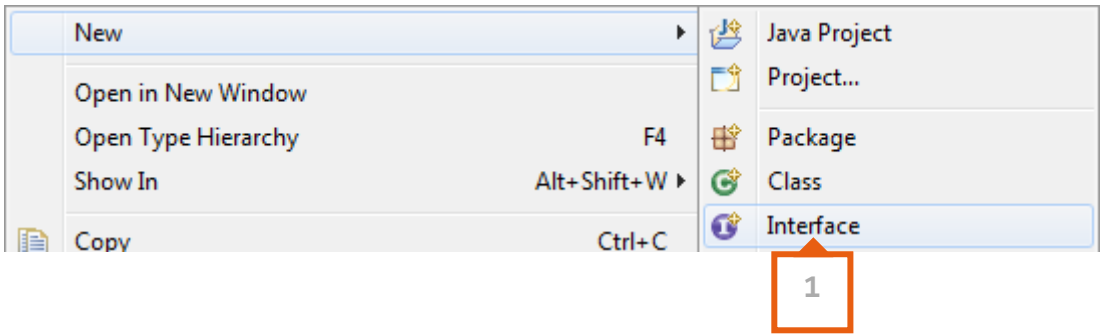
Tenha em consideração o diagrama seguinte:



O diagrama mostra o esquema de classes que foi usado nos módulos anteriores acrescido de duas entidades: a classe Barco e a classe Pessoa. Todas as classes vistas neste diagrama têm ações em comum ainda que não estejam hierarquicamente relacionadas, por exemplo todas as entidades são móveis. Seria útil num programa poder pegar em todos estes objetos de forma uniforme e invocar um método `mover`, de forma semelhante ao que foi feito no módulo de polimorfismo. Mas isto não é possível pois eles não derivam da mesma classe. Para este fim utilizamos a interface, que funciona como um protocolo que todos implementam.

- **Crie** o package **interfaces**

- Agora **crie** uma **Interface** 1



- **Atribua** o nome **movel** à interface e **clique** em **Finish**

- **Introduza** o seguinte código:

```
package interfaces;
public interface movel {
    public void mover();
    public int obterVelocidade();
}
```

Acabámos de definir duas ações para esta interface. Todas as classes que cumpram este protocolo têm que ter estas duas ações definidas. A palavra reservada para a utilização de interfaces é **implements** e a sintaxe é a seguinte:

```
public class Classe1 implements Interface1 {
```

Vamos colocar este conceito em prática na classe `Carro`.

› **Altere** a declaração da classe `Carro.java` para:

```
public class Carro implements movel{
```

Repare como, neste momento, encontra um erro na classe `Carro`.

Quando referenciamos classes que não se encontram no mesmo package é-nos indicado um erro. Para que isto não aconteça é necessário importar as classes ou interfaces de um determinado package.

› **Adicione** o código que se encontra destacado à classe `Carro`:

```
package carro;
import interfaces.movel;

public class Carro implements movel{
```

Após esta alteração o Eclipse já reconhece a interface `movel` nesta classe. No entanto, o erro persiste pois esta classe não segue ainda o protocolo. Precisamos então de especificar os métodos `mover` e `obterVelocidade` na classe `Carro` para que esta cumpra o protocolo estabelecido pela interface.

Note que o método `obterVelocidade` já está implementado exatamente com as especificações que a interface define.

› **Implemente** o método `mover` na classe `Carro` através do seguinte código:

```
/**
* Faz o carro mover-se, aumentando a sua velocidade corrente
*/
```

```
public void mover() {
    acelerar(); // chamada ao acelerar sem parametros que aumenta 10km/h
}
```

Para aumentar um pouco a complexidade do programa, vamos contemplar também a classe `Barco`.

- **Crie** a classe `Barco.java` no package **interfaces**

- Nessa mesma classe **adicione** o código:

```
package interfaces;

public class Barco implements movel{
    private int velocidade_corrente, numvelas;
    private boolean motorLigado;

    public Barco(int numvelasbarco){
        numvelas = numvelasbarco;
        velocidade_corrente = 0;
        motorLigado = false;
    }

    public void ligarMotor(){
        motorLigado = true;
    }

    public void desligarMotor(){
        motorLigado = false;
    }

    /**
```

```
     * Aumenta a velocidade do barco em 5km/h
     */
    public void mover() {
        velocidade_corrente +=5;
    }

    public int obterVelocidade() {
        return velocidade_corrente;
    }

    /**
     * Mostra as informacoes do barco
     */
    public void mostrar(){
        System.out.println("");
        System.out.println("Barco: ");
        System.out.println("Numero de velas -> " + numvelas);
        System.out.println("Estado do motor -> " +
            ((motorLigado == true)? "ligado":"desligado"));
        System.out.println("Velocidade -> " + velocidade_corrente);
    }
}
```

- **Crie** também a classe `TesteInterfaces.java` no package **interfaces** e **inclua** o seguinte código:

```
package interfaces;

import carro.Carro;

public class TesteInterfaces {
    public static void main(String[] args) {
        Carro carro1 = new Carro("35-12-FL","Audi");
        carro1.mover();
```

```
        carro1.mostrar();

        Barco barco1 = new Barco(2);
        barco1.mover();
        barco1.mostrar();
    }
}
```

- **Compile** e **teste** a classe

Repare como através do mesmo método mover, tanto o barco como o carro aumentaram a sua velocidade. No entanto, esta sintaxe não é muito útil porque, sendo de tipos diferentes, não poderíamos ter ambos os objetos num array e pedir que todos se movessem. Para isto é preciso ver estes objetos como do tipo da Interface.

- **Redefina** o método `main` da classe `TesteInterfaces` tendo em conta o seguinte código:

```
package interfaces;

import carro.Carro;

public class TesteInterfaces {
    public static void main(String[] args) {
        movel[] objetosmoveis = new movel[2];

        objetosmoveis[0] = new Carro("35-12-FL","Audi");
        objetosmoveis[1] = new Barco(2);

        for (int i = 0;i < objetosmoveis.length;++i){
            objetosmoveis[i].mover();
            System.out.println("Velocidade após mover :" + objetosmoveis[i].obterVelocidade());
        }
```

```
    }
}
```

› **Teste** esta classe

A interface movel funciona como um tipo. É possível ter um objeto do tipo "Carro" como um objeto do tipo "movel", através da seguinte instrução:

```
movel m = new Carro("11-11-xx","Teste");
```

A vantagem desta sintaxe é que se torna possível fazer isto para todas as classes que implementam a interface. Assim, podemos ver todos esses objetos como sendo desse tipo, o que permite invocar um método da interface, com se vê na linha abaixo:

```
m.mover();
```

Com esta notação conseguimos criar arrays com objetos diferentes, desde que estes implementem a interface `movel`. Percorrendo este array é possível fazer com que todos os objetos se movam ou obter a sua velocidade, independentemente do seu tipo.

⚠ Num array de objetos de um tipo definido por uma interface, não é possível executar métodos específicos da classe do objeto. Ou seja, no código acima, a instrução `m.obterNumeroRodas()` seria inválida pois trata-se de uma ação específica da classe Carro que não consta na interface `movel`.