15) Ao instanciar o objeto temos que passar os parâmetros do construtor.

```
 Pessoa pessoa = new Pessoa("Marcos", 10, 1.75f);
```

16) Podemos também, sobrecarregar o método gerando um tipo de polimorfismo.

```
.....
public Pessoa(String nome, int idade, float altura) {
        this.nome = nome;
        this.idade = idade;
        this.altura = altura;
    }

    public Pessoa(String nome, int idade) {
        this.nome = nome;
        this.idade = idade;

    }
    public Pessoa() {

    }
.....
```

17) Vamos avançar e criar duas classes, um chamadas Musico e a outra Ator.

```
public class Musico {
    static final int TEMPODEVIDA = 100;
    private String nome;
    private int idade;
    private float altura;
    private  String escolaMusica;
}
```

```
public class Ator {
    static final int TEMPODEVIDA = 100;
    private String nome;
    private int idade;
    private float altura;
    private  String escolaAtor;
    }
```

18) Se observar, temos muitos valos repetidos em ambas na classe. Podemos neste caso, refatorar e realizar o reaproveitamento de códigos. Vamos iniciar este processo transformando a classe “Pessoa” em uma classe abstrata. Você não poderá mais instanciar um objeto Pessoa diretamente, apenas por meio e uma classe concreta.

```
....
public abstract class Pessoa {
....
```

19) Agora você deve implementar mais um pilar da Orientação a objeto que é a herança, para isso adicione extends após o nome da classe e chame classe Pessoa (Agora temos a ação que a subclasse(Classe filha) herda da superclasse(Classe mão/pai). O método

recebe a annotation @Override pois está subscrevendo o método abstrato da classe Pessoa.

```
.....
public class Musico extends Pessoa {
    private  String escolaMusica;
    private List<String> musica;

    public Musico(String nome, int idade, float altura, String escolaM
usica) {
        super(nome, idade, altura);
        this.escolaMusica = escolaMusica;

    }

   @Override
    public int returneTempoVida() {
         return (TEMPODEVIDA - getIdade()) / 2;
    }
......
```

20) Repita a ação anterior para a classe “Ator”.

```
public class Ator extends Pessoa {
        private  String escolaAtor;
    private List<String> filme;
    public Ator(String nome, int idade, float altura, String escolaAto
r) {
        super(nome, idade, altura);
        this.escolaAtor = escolaAtor;
    }

    @Override
    public int returneTempoVida() {
        return (TEMPODEVIDA - getIdade()) / 2;
    }
```

21) Obs: Ao criar um método abstrato ele não é implementado e é sobescrito na classe concreta.

Implementação do método da subclasse(classe concreta)

```
@Override
    public int returneTempoVida() {
         return (TEMPODEVIDA - getIdade()) / 2;
    }
```

Método Abstrato

```
public abstract int returneTempoVida();
```

22) Uma interface necessita implementar todos os métodos e atributos. Crie uma interfaces chamada “Interfaces”. Nesta interface ela obriga a classe a implementar o método returneTempoVida.

```
public interface Interfaces {
    int returneTempoVida();
}
```

23) Vamos criar uma classe chamada Dançarino que herda de Pessoa e implementa a interface.

```
.....
public class Dancarino extends Pessoa implements Interfaces{

public Dancarino(String nome, int idade, float altura) {
        super(nome, idade, altura);

    }

    @Override
    public int returneTempoVida() {
        return (TEMPODEVIDA - getIdade());
    }
....
```

24) Agora é só testar as classes.

‘

Até aqui tudo ok!?

Agora vamos para próxima situação de aprendizagem.