Abstração

É utilizada para a definição de entidades do mundo real. Sendo onde são criadas as classes. Essas entidades são consideradas tudo que é real, tendo como consideração as suas características e ações

Entidade	Características	Ações
Carro, Moto	tamanho, cor, peso, altura	acelerar, parar, ligar, desligar
Elevador	tamanho, peso máximo	subir, descer, escolher andar
Conta Banco	saldo, limite, número	depositar, sacar, ver extrato

Abstração

Uma classe é reconhecida quando tem a palavra reservada “class”. Na imagem ao lado é mostrado a classe “Conta” com seus atributos (características) e métodos (ações).

```java
public class Conta {
    int numero;
    double saldo;
    double limite;

    void depositar(double valor){
        this.saldo += valor;
    }

    void sacar(double valor){
        this.saldo -= valor;
    }

    void imprimeExtrato(){
        System.out.println("Saldo: "+this.saldo);
    }
}
```

Encapsulamento

É a técnica utilizada para esconder uma ideia, ou seja, não expor detalhes internos para o usuário, tornando partes do sistema mais independentes possível. Por exemplo, quando um controle remoto estraga apenas é trocado ou consertado o controle e não a televisão inteira. Nesse exemplo do controle remoto, acontece a forma clássica de encapsulamento, pois quando o usuário muda de canal não se sabe que programação acontece entre a televisão e o controle para efetuar tal ação.

Encapsulamento

Em um processo de encapsulamento os atributos das classes são do tipo private. Para acessar esses tipos de modificadores, é necessário criar métodos setters e getters.
Por entendimento os métodos setters servem para alterar a informação de uma propriedade de um objeto. E os métodos getters para retornar o valor dessa propriedade.
Veja um exemplo de encapsulamento, no exemplo ao lado gera-se os atributos privados (private) e é realizado o processo de geração dos métodos setters e getters.

Métodos getters	Métodos setters

public String getNome() {
return nome;
}
	public void setNome(String nome) {
	this.nome = nome;
	}

public double getSalario() {
return salario;
}
	public void setSalario(double salario) {
	this.salario = salario;
	}

Encapsulamento

```java
public class Funcionario {
    private double salario;
    private String nome;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public void setSalario(double salario) {
        this.salario = salario;
    }

    public double getSalario() {
        return salario;
    }
}
```

No exemplo ao lado, é instanciado a classe Funcionario, onde a variável de referência é usada para invocar os métodos setters, informando algum dado. Ao final, é usado os métodos getters dentro do System.out.println para gerar a saída dos resultados que foram passados nos métodos setters

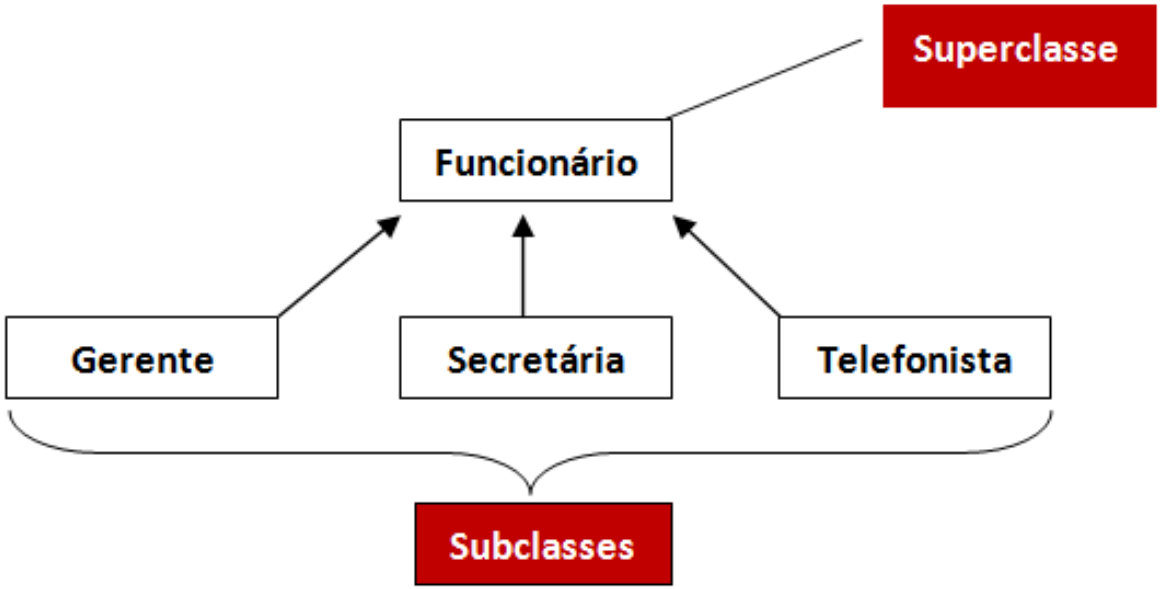
Encapsulamento

```
1  public class TestaFuncionario {
2
3      public static void main(String[] args) {
4          Funcionario funcionario = new Funcionario();
5          funcionario.setNome("Thiago");
6          funcionario.setSalario(2500);
7
8          System.out.println(funcionario.getNome());
9          System.out.println(funcionario.getSalario());
10
11     }
12 }
```

Herança

Na Programação Orientada a Objetos o significado de herança tem o mesmo significado para o mundo real. Assim como um filho pode herdar alguma característica do pai, na Orientação a Objetos é permitido que uma classe herde atributos e métodos da outra, tendo apenas uma restrição para a herança. Os modificadores de acessos das classes, métodos e atributos só podem estar com visibilidade public e protected para que sejam herdados.

Uma das grandes vantagens de usar o recurso da herança é na reutilização do código. Esse reaproveitamento pode ser acionado quando se identifica que o atributo ou método de uma classe será igual para as outras. Para efetuar uma herança de uma classe é utilizada a palavra reservada chamada extends.



Herança

Para saber se estamos aplicando a herança corretamente, realiza-se o teste “É UM”. Esse teste simples ajuda a detectar se a subclasse pode herdar a superclasse.

Por exemplo, na Figura 3, está mostrando que a classe Gerente herda da classe Funcionário, se for aplicado o teste “É UM” nota-se que o teste é aprovado, pois o Gerente também “É UM” Funcionário.

Veja nos exemplos abaixo como aplicar o recurso da herança em uma classe.

Na imagem ao lado, existe a superclasse Funcionario que servirá de base para as subclasses usarem seus atributos ou métodos.

```java
public class Funcionario {
    private String nome;
    private double salario;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public double getSalario() {
        return salario;
    }

    public void setSalario(double salario) {
        this.salario = salario;
    }

    public double calculaBonificacao(){
        return this.salario * 0.1;
    }

}
```

Herança

```java
public class Gerente extends Funcionario {
    private String usuario;
    private String senha;

    public String getUsuario() {
        return usuario;
    }

    public void setUsuario(String usuario) {
        this.usuario = usuario;
    }

    public String getSenha() {
        return senha;
    }

    public void setSenha(String senha) {
        this.senha = senha;
    }

    public  double calculaBonificacao(){
        return this.getSalario() * 0.6 + 100;
    }

}
```

No exemplo ao lado, podemos ver que a classe Gerente está herdando da classe Funcionario através da palavra reservada extends. Acontece também a mudança do comportamento de herança, a partir do método calculaBonificacao que é sobrescrito, pois entende-se que o valor da classe Gerente é diferente para as demais.

Herança

```
1  public class Secretaria extends Funcionario {
2      private int ramal;
3
4      public void setRamal(int ramal) {
5          this.ramal = ramal;
6      }
7
8      public int getRamal() {
9          return ramal;
10     }
11 }
```

Secretaria

Herança

```java
public class Telefonista extends Funcionario {
    private int estacaoDeTrabalho;

    public void setEstacaoDeTrabalho(int estacaoDeTrabalho) {
        this.estacaoDeTrabalho = estacaoDeTrabalho;
    }

    public int getEstacaoDeTrabalho() {
        return estacaoDeTrabalho;
    }
}
```

Telefonista

Herança

No próximo exemplo são apresentados todas as classes e mostrada a reutilização de código, um exemplo são os atributos nome e salario. Portanto, não foi preciso criar em todas as classes, apenas criou-se na superclasse. Apenas lembrando que o acesso dos atributos ou métodos de uma superclasse é permitido somente se estão definidos com o modo de visibilidade como public ou protected.

```
1  public class TestaFuncionario {
2
3      public static void main(String[] args) {
4
5          Gerente gerente = new Gerente();
6          gerente.setNome("Carlos Vieira");
7          gerente.setSalario(3000.58);
8          gerente.setUsuario("carlos.vieira");
9          gerente.setSenha("5523");
10
11         Funcionario funcionario = new Funcionario();
12         funcionario.setNome("Pedro Castelo");
13         funcionario.setSalario(1500);
14
15         Telefonista telefonista = new Telefonista();
16         telefonista.setNome("Luana Brana");
17         telefonista.setSalario(1300.00);
18         telefonista.setEstacaoDeTrabalho(20);
```

Herança

```
19
20          Secretaria secretaria = new Secretaria();
21          secretaria.setNome("Maria Ribeiro");
22          secretaria.setSalario(1125.25);
23          secretaria.setRamal(5);
24
25          System.out.println("##### Gerente #####");
26          System.out.println("Nome.: "+gerente.getNome());
27          System.out.println("Salário.: "+gerente.getSalario());
28          System.out.println("Usuário.: "+gerente.getUsuario());
29          System.out.println("Senha.: "+gerente.getSenha());
30          System.out.println("Bonificação.: "+gerente.calculaBonificacao());
31          System.out.println();
32
33          System.out.println("##### Funcionário #####");
34          System.out.println("Nome.: "+funcionario.getNome());
35          System.out.println("Salário.: "+funcionario.getSalario());
36          System.out.println("Bonificação.: "+funcionario.calculaBonificacao());
37          System.out.println();
38
```

Herança

```
38
39          System.out.println("##### Telefonista #####");
40          System.out.println("Nome.: "+telefonista.getNome());
41          System.out.println("Salário.: "+telefonista.getSalario());
42          System.out.println("Estação de Trabalho.: "+telefonista.getEstacaoDeTrabalho());
43          System.out.println("Bonificação.: "+telefonista.calculaBonificacao());
44          System.out.println();
45
46          System.out.println("##### Secretária #####");
47          System.out.println("Nome.: "+secretaria.getNome());
48          System.out.println("Salário.: "+secretaria.getSalario());
49          System.out.println("Ramal.: "+secretaria.getRamal());
50          System.out.println("Bonificação.: "+secretaria.calculaBonificacao());
51          System.out.println();
52      }
53
54  }
```