a) POO é uma forma de modelagem de dados ligada a criação de objetos, métodos e atributos que tente criar o mais próximo do real um objeto virtual com abstração de certas propriedados desse objeto real. Pilares são abstração, encapsulamento, herança e polimorfismo.

b) Abstração diz respeito a como recriar esse objeto de forma a definir sua importância dentro do nosso código, o que ele irá fazer e como fazer, criar uma classe de forma que não entre em conflito com outra classe e que apenas esse objeto em questão seja responsável por tal tarefa e execute de forma desejada e clara. Exemplo de classe abaixo é responsável apenas por fazer cálculos simples e pode ser usado em outras classes ou programas, o que define ela é a propriedade de ser uma calculadora simples e funciona mais ou menos como uma calculadora real(objeto) de forma mais abstrata.

class Calculadora{

public int somar(int a, int b){

return a + b;

}

public int subtrair(int a, int b){

return a - b;

}

}

c) Encapsulamento, como o próprio nome diz, é o pilar da orientação a objetos que mexe com o isolamento de certas propriedes privadas de uma classe, isolar certos valores e atributos de uma classe de forma que apenas seja permitido manipular o que o desenvolvedor/usuário permitir. Criar métodos que apenas exponha o necessário e o que pode ser modificado é uma forma segura, como por exemplo as funções get and set.

public class Cofre{

private int dinheiro = 0;

public int getDinheiro(){

return dinheiro;

}

Public int setDeposito(int dinheiro){

this.dinheiro = dinheiro;

}

}

d) Herança, é simples, um objeto pode herdar as propriedades de outra classe, por padrão, essa classe funciona como uma cópia da que passou a herança, ela pode conter melhorias/modificações que a distigue da classe pai, porém, podem fazer uso dos mesmos atributos e métodos. Abaixo o exemploi da classe que herda a cor dos olhos da mãe(propriedade) e no construtor da classe filha o método é que informa a cor dos olhos é chamado(Comportamento).

abstract class Mae{

private String corDosOlhos = “Tenho olhos azuis”;

public void corDosOlhos(){

System.out.println(corDosOlhos);

}

}

public class Filha extends Mae{

public Filha(){

this.corDosOlhos();

}

}

e)Como o próprio nome fala, várias formas, o comportamento herdade de uma classe precisa ser modificado em algumas implementações, é necessário dar formas e mudanças necessárias em certos métodos e nas classes que herdam da classe mãe, dessa forma, usar o Polimorfismo é essencial para que o código seja dinâmico e atenda as específicações do desenvolvedor quando for preciso. No exemplo abaixo a classe filha herda a cor dos olhos da mãe, chamando o método padrão da mãe no construtor, porém, se caso a filha herdou os olhos do pai, ela tem a cor dos olhos diferentes, ou seja, sobrescrevi o métod da mãe para atender as modificações necessárias quando foi preciso.

abstract class Mae{

private String corDosOlhos = "Tenho olhos azuis";

public void cor(String olho){

corDosOlhos = olho;

}

public void getCorDosOlhos(){

System.out.println(corDosOlhos);

}

}

class Filha extends Mae{

private String herancaPaterna = "Tenho olhos castanhos";

public Filha(){

super.getCorDosOlhos();

}

@Override

public void getCorDosOlhos() {

System.out.println(herancaPaterna);

}

}

f) Representar um objeto real de forma simples, atrabuir comportamentos únicos e específicos em uma classe, segurança ao usar formas limpas e eficientes de encapsulamento, fácil de manter e modificar uma classe se necessário, evitar colisões, conflitos e passar comportamentos para outras classes quando preciso .