Java:

Documentation: https://docs.oracle.com/javase/tutorial/

Como trocar a versão do java em uso: https://youtu.be/-ecG6B_jNW8?si=P73tbRAcdk9GsQYa

Datas em Java:

-Datas para o mundo todo:
ISO 8601

LocalDate date = LocalDate.now();, isso me retorna a data do dia atual da forma ano/mês/dia.

- date.getDayOfWeek(); , retorna o dia da semana dessa data em inglês.
- Para traduzir eu coloco junto dos métodos o .getDisplayName(TextStyle.FULL, new Locale("pt","BR"));

  Obs:É necessário o import da classe Locale, sendo que posso criar a variável antes.

- LocalDateTime é uma classe usada para retornar o tempo e conjunto da data.
- Use LocalDateTime agora = LocalDateTime.now(); para retornar a data no exato momento
- Métodos para obter informações dessa data: agora.getHour(), getSeconds(),…

POO:

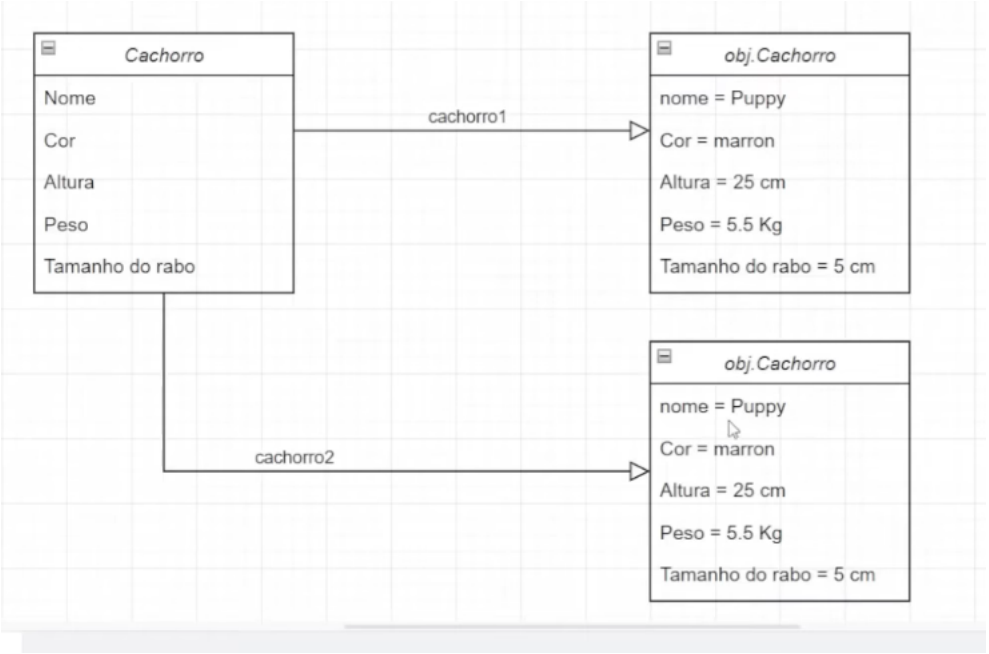
-byte: -128 até 127
-short: -32768 até 32767
-char: 0 a 65535
-int: -2 bilhões a 2 bilhões
-long: -9 trilhões a 9 trilhões

- Encapsulamento:

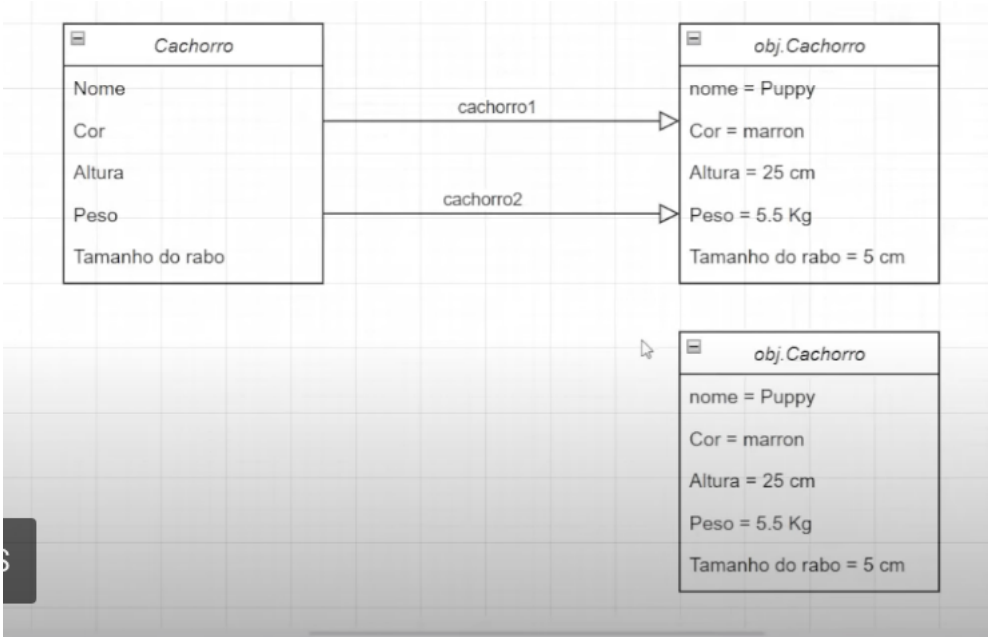
Visibilidade	public	protected	default	private
A partir da mesma classe	√	√	√	√
Qualquer classe no mesmo pacote	√	√	√	×
Qualquer classe filha no mesmo pacote	√	√	√	×
Qualquer classe filha em pacote diferente	√	√	×	×
Qualquer classe em pacote diferente	√	×	×	×

-default: ele não precisa ser escrito, ele está presente quando não digito nada.

- Modelagem Orientada a Objetos:



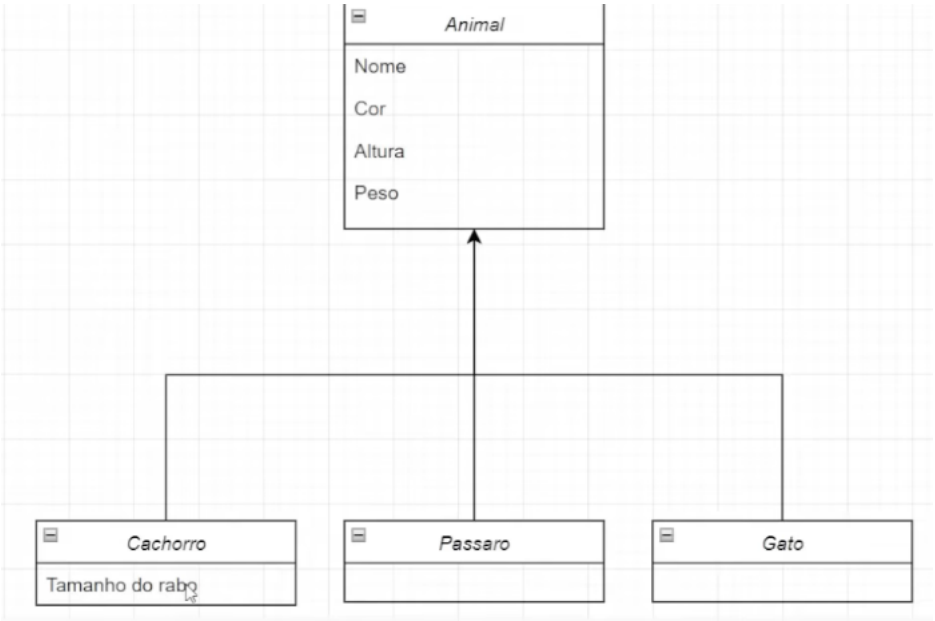
-Cada objeto tem um espaço específico na memória.
-Porém quando eu faço cachorro2 = cachorro1, eu aloco o cachorro2 ao mesmo endereço de memória do cachorro1.



-O que ocorre após o “cachorro2=cachorro1”, é o que está na imagem acima, então o segundo objeto se perde na memória e assim o garbage colector do Java, pega e limpa da memória esse objeto, sendo perdido e não podendo ser acessado novamente dessa forma.

-Contar números de objetos em uma classe: é só usar uma variável static.

- Herança:



-Terei uma classe principal, ex: Animal com atributos e métodos específicos, e nas classes que serão filhas dessa eu vou usar o **public class Cachorro extends Animal.** Assim eu faço a classe cachorro herdar a classe animal.

-Na classe Animal: eu preciso declarar os atributos como protected, pois assim as classes filho podem acessar esses atributos, além da necessidade de criar os métodos Getters and Setters.

-Nas classes que herdam: como cachorro, gato, etc, eu uso em seus construtores o super();, passando as entradas do construtor nessa linha de código.
Ex:

```
public Gato(String nome, String cor, double peso) {
        super(nome, cor, peso);
        this.nome = nome;
        ...
}
```

-Usando o @Override: Eu uso o Override para modificar um método da classe pai na classe filho, assim aquela classe terá um método diferente, ou seja eu herdo o método fazerBarulho(); da classe pai, mas para cada animal eu farei um barulho diferente, então eu uso o Override para mudar isso.

• Polimorfismo:

-Definimos Polimorfismo como um princípio a partir do qual as classes derivadas de uma única classe base são capazes de invocar os métodos que, embora apresentem a mesma assinatura, comportam-se de maneira diferente para cada uma das classes derivadas.

-O Polimorfismo é um mecanismo por meio do qual selecionamos as funcionalidades utilizadas de forma dinâmica por um programa no decorrer de sua execução.

-Com o Polimorfismo, os mesmos atributos e objetos podem ser utilizados em objetos distintos, porém, com implementações lógicas diferentes.

-Por exemplo: podemos assumir que uma bola de futebol e uma camisa da seleção brasileira são artigos esportivos, mais que o cálculo deles em uma venda é calculado de formas diferentes.

-Outro exemplo: podemos dizer que uma classe chamada Vendedor e outra chamada Diretor podem ter como base uma classe chamada Pessoa, com um método chamado CalcularVendas.

Se este método (definido na classe base) se comportar de maneira diferente para as chamadas feitas a partir de uma instância de Vendedor e para as chamadas feitas a partir de uma instância de Diretor, ele será considerado um método polimórfico, ou seja, um método de várias formas.

-Assim podemos ter na classe base o método CalcularVendas:

```
public decimal CalcularVendas(){

    decimal valorUnitario = decimal.MinValue;

    decimal produtosVendidos = decimal.MinValue;

    return valorUnitario * produtosVendidos;

 }
```

-Na classe Vendedor temos o mesmo método, mais com a codificação diferente:

```
public decimal CalcularVendas(){

    decimal valorUnitario = 50;

    decimal produtosVendidos = 1500;

    return valorUnitario * produtosVendidos;

 }
```

-O mesmo ocorre na classe Diretor:

```
public decimal CalcularVendas() {

    decimal valorUnitario = 150;

    decimal produtosVendidos = 3800;

    decimal taxaAdicional = 100;

    return taxaAdicional + (valorUnitario * produtosVendidos);

 }
```

-Classes Abstratas para controle do Polimorfismo: Seriam classes que vai ser pai de outras classes, porém ela não pode ser instanciada sozinha, ela só pode ser herdada, basicamente ela dita as regras, ela fala o que toda classe que herda ela deve ter.

-Para criar uma classe abstrata é só fazer: *public abstract class Animal{}.*

-Métodos Abstratos: Dentro da Classe abstrata, caso eu queira criar um método abstrato, ou seja, e determine que cada animal tem que implementar o método do seu próprio jeito eu uso: *public abstract void fazerBarulho();.*

• Complementações:

-Comente seus códigos, explicando cada linha usando ou // ou /* */.

-Valores default de variáveis:

```java
int inteiro; //int, long, byte = 0;
float $$; //double e float = 0.0
boolean $87; //boolean tem como default false
char ch; // char é vazio
String teste; // valor default de qualquer objeto é null
```

-Casting em java: Transformações de tipos de variáveis:

```java
double d = 5.5d;
float f = 3.00f;

float x = f + (float) d;
System.out.println(x);
```

-Casting possíveis:

CASTINGS POSSÍVEIS

Abaixo estão relacionados todos os casts possíveis na linguagem Java, mostrando a conversão **de** um valor **para** outro. A indicação **Impl.** quer dizer que aquele cast é implícito e automático, ou seja, você não precisa indicar o cast explicitamente (lembrando que o tipo boolean não pode ser convertido para nenhum outro tipo).

PARA: / DE:	byte	short	char	int	long	float	double
byte	----	*Impl.*	(char)	*Impl.*	*Impl.*	*Impl.*	*Impl.*
short	(byte)	----	(char)	*Impl.*	*Impl.*	*Impl.*	*Impl.*
char	(byte)	(short)	----	*Impl.*	*Impl.*	*Impl.*	*Impl.*
int	(byte)	(short)	(char)	----	*Impl.*	*Impl.*	*Impl.*
long	(byte)	(short)	(char)	(int)	----	*Impl.*	*Impl.*
float	(byte)	(short)	(char)	(int)	(long)	----	*Impl.*
double	(byte)	(short)	(char)	(int)	(long)	(float)	----

-Organização de valores: pode usar "_" em números para entender melhor suas unidades:

```java
int a = 23_45_67_89;
```

-Palavras chave do Java:

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

-Garbage Colector: Serve para limpar coisas da memória, como objetos, que não vão ser ou não estão sendo utilizados enquanto o seu código funciona.

-Varargs: quando eu não tenho a informação de quantas variáveis daquela eu irei receber no método assim eu uso o varargs :

public somar(int… numeros){}.