

# Aula 9

# Falta pouco...

## Vamos aprender hoje:

- Datas
  - Date
  - Gregorian Calendar
  - LocalDate e LocalDateTime
- Estrutura Enum;
- Arquivos;
- Banco de dados – JDBC;
- Classpath;
- Jars;
- Bibliotecas;

# Lidando com Datas e Horas

## Date

- Podemos criar uma variável date assim:

```
Date date = new Date();
```

```
System.out.println(date);
```

*Saída: Sat Aug 08 16:48:42 BRT 2020*

- Na criação do objeto Date, é capturada a Data/Hora do relógio do computador;
- Mas como colocar uma data digitada pelo usuário em uma variável Date?
- Temos que usar conversores pois a data pode estar em vários formatos;
- A classe SimpleDateFormat nos servirá;

# Lidando com Datas e Horas

## Data - SimpleDateFormat

```
SimpleDateFormat simpleDateFormat = new SimpleDateFormat("dd/MM/yyyy");  
Date date = new Date();  
System.out.println("Date antes de formatar: " + date);  
System.out.println("Date depois de formatar: " + simpleDateFormat.format(date));
```

O contrário também é importante: mostrar a data em forma de texto;

```
//Podemos usar outros formatos no construtor do conversor  
SimpleDateFormat simpleDateFormat = new SimpleDateFormat("dd/MM/yyyy");  
String stx = "07/11/1978";  
Date dataUsuario = simpleDateFormat.parse(stx);  
System.out.println("Date digitado e formatado: " + dataUsuario);
```

# Lidando com Datas e Horas

## O formato do formatter

`dd`: dia do mês com dois dígitos (01 à 31)

`MM`: mês do ano com dois dígitos (01 à 12)

`yyyy`: ano com quatro dígitos (exemplo: 2018)

`HH`: horas com dois dígitos (00 à 24)

`mm`: minutos com dois dígitos (00 à 59)

`ss`: segundos com dois dígitos (00 à 59)

`hh`: horas (até 12) com dois dígitos (00 à 12)

# Lidando com Datas e Horas

## Gregorian Calendar

- A classe Date é bem simples e serve para armazenar datas apenas.
- Ainda temos que manipular as datas não é?
- Por exemplo acrescentar 10 dias, verificar qual o dia da semana
- Para isso vamos usar a classe GregorianCalendar.
- Assim como o Date, quando instanciado é capturada a Data/Hora do relógio do computador;

# Lidando com Datas e Horas

## Gregorian Calendar

```
GregorianCalendar calendar = new GregorianCalendar();  
//Mostra qual o dia da semana 1 = domingo, 2=segunda, etc  
System.out.println(gc.get(gc.DAY_OF_WEEK));  
//Adiciona 2 meses à data atual  
System.out.println( gc.add(gc.MONTH, 2));  
//Imprime falso. 2009 não é bissexto.  
System.out.println(gc.isLeapYear(2009));  
//Atribui a data do GregorianCalendar à uma variável Date  
Date d1 = gc.getTime();  
//Armazena a data de d1 para o GregorianCalendar gc.  
gc.setTime(d1);
```

# Lidando com Datas e Horas

## LocalDate

- O LocalDate usa outra classe como formatador o DateTimeFormatter

```
LocalDate hoje = LocalDate.now();  
System.out.println("LocalDate antes de formatar: " + hoje);  
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");  
String hojeFormatado = hoje.format(formatter);  
System.out.println("LocalDate depois de formatar: " + hojeFormatado);
```



# Lidando com Datas e Horas

## LocalDateTime

O LocalDateTime usa outra classe como formatador o `DateTimeFormatter`

```
LocalDateTime agora = LocalDateTime.now();  
System.out.println("LocalDateTime antes de formatar: " + agora);  
formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss");  
String agoraFormatado = agora.format(formatter);  
System.out.println("LocalDateTime depois de formatar: " + agoraFormatado);
```

## Criação de estruturas de dados organizados

As instâncias dos tipos **enum** são criadas e nomeadas junto com a declaração da classe, sendo fixas e imutáveis (o valor é fixo);

Não é permitido criar novas instâncias com a palavra chave **new**;

O construtor é declarado **private**, embora não precise de modificador explícito;

Seguindo a convenção, por serem objetos constantes e imutáveis (**static final**), os nomes declarados recebem todas as letras em MAIÚSCULAS;

As instâncias dos tipos **enum** devem obrigatoriamente ter apenas um nome;

**Opcionalmente**, a declaração da classe pode incluir variáveis de instância, construtor, métodos de instância, de classe, etc.

# Estrutura Enum

## Criação de estruturas de dados organizados

- São tipos de campos que consistem em um conjunto fixo de constantes (**static final**),
- Como uma lista de valores pré-definidos.
- Pode ser definido um tipo de enumeração com a palavra chave **enum**;
- Todos os tipos enums implicitamente estendem a classe `java.lang.Enum`;
- O Java não suporta herança múltipla, não podendo estender nenhuma outra classe.

# Estrutura Enum

## Exemplo

```
public enum PessoasEnum {  
    GERENTE,  
    DIRETOR,  
    PRESIDENTE,  
    CLIENTE;  
}
```

# Estrutura Enum

## Exemplo

```
public class TestadoraEnum {  
    public static void main(String[] args) {  
        String testaEnum = "GERENTE";  
        PessoasEnum presidente = PessoasEnum.PRESIDENTE;  
        PessoasEnum gerente = PessoasEnum.GERENTE;  
  
        if( testaEnum.equalsIgnoreCase(PessoasEnum.PRESIDENTE.name()) ) {  
            System.out.println("Tipo da Pessoa = " + presidente.name());  
        }  
        else if( testaEnum.equalsIgnoreCase(PessoasEnum.GERENTE.name()) ) {  
            System.out.println("Tipo da Pessoa = " + gerente.name());  
        }  
    }  
}
```

# Arquivos – Leitura e Escrita

Existem classes que fazem esse trabalho:

- BufferedReader, FileReader e BufferedWriter e FileWriter;
- Fazem a leitura do arquivo, linha a linha;
- O que define a quebra de linha?
- Após lida basta manipular a linha com os recursos da String;
- Algumas regras importantes:
  - Passar o caminho do arquivo corretamente;
  - Fechar o arquivo no final da leitura ou da escrita;

## Exemplo de leitura

```
public static void leitor(String path) throws IOException {  
  
    BufferedReader buffRead = new BufferedReader(new FileReader(path));  
    String linha = "";  
  
    while (true) {  
        linha = buffRead.readLine();  
        if (linha != null) {  
            System.out.println(linha);  
        }  
        else  
            break;  
    }  
    buffRead.close();  
}
```

## Exemplo de escrita

```
public static void escritor(String path) throws IOException {  
    BufferedWriter buffWrite = new BufferedWriter(new FileWriter(path));  
    String linha = "";  
    Scanner in = new Scanner(System.in);  
    System.out.println("Escreva algo: ");  
    linha = in.nextLine();  
    buffWrite.append(linha + "\n");  
    buffWrite.close();  
}
```



## JDBC - Java Database Connectivity

- É uma **API** de classes e interfaces em Java;
- Possibilita conectar-se através de um **driver** em um banco de dados;
- Cada banco de dados possui um driver específico;
- O **driver** executa instruções SQL no banco de dados relacional.
- Para ocorrer a comunicação entre a aplicação e o SGBDs é necessário possuir um driver para a conexão desejada.
- As empresas de SGBDs oferecem o driver de conexão que seguem a especificação JDBC para uso de desenvolvedor.

# Classpath

## Classpath

- O que é Classpath?
- Vamos pensar sobre o termo;
- Class + path;
- O que parece para vocês?
- É um parâmetro, que indica à Máquina Virtual Java (JVM) onde procurar pacotes e bibliotecas;
- É definido pelo usuário, e pode conter vários caminhos;
- As bibliotecas do Java são arquivos .class e os pacotes .jar, .ear e .war;

# Jar, War e Ear - O que é isso?

- Java é tão popular, tão poderosa e porquê ela não faz um arquivo executável (.exe) para rodar os programas feitos por ela?
- Esta pergunta é frequente principalmente por pessoas que migram para Java acostumadas a usar C, C++, VB, e Delphi (pascal).
- Porque o programa Java não gera um executável?
- Java é poderosa não somente por contar com os princípios da orientação a objeto, mas também pela **portabilidade**.
- Os códigos são portáveis (multi-plataforma), por isso o compilador não cria arquivo executável, pois isso acabaria com a portabilidade.

# Jar, War e Ear – Pacotes do Java

## Conceito de pacotes

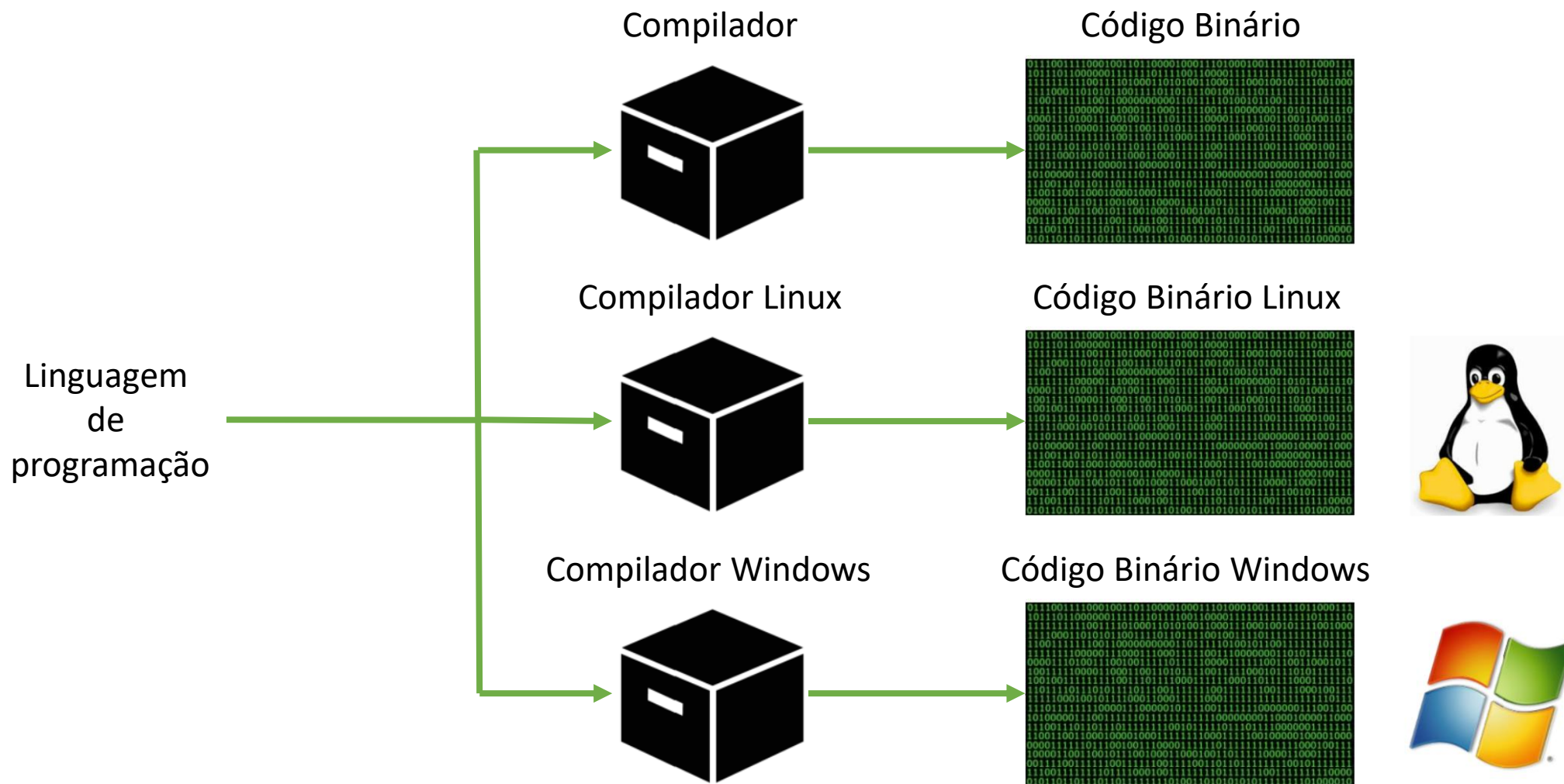
- As classes em Java podem ser empacotadas para execução em outra plataforma;
- Jar - Pacote de classes que pode definir uma biblioteca ou um programa de desktop ~~executável~~ (interpretável).
- War – O mesmo que o Jar mas para interpretação em ambientes web
  - Podem conter arquivos do tipo JAR, HTML, CSS, Imagens e etc.
  - É o mais utilizado para aplicações web mais simples, ou de baixa complexidade. (Servidores web como: Apache)
- Ear - O Pacote para web corporativa, que pode conter War e Ejb.
  - Esse pacote e utilizado para aplicações mais complexas
  - Necessitam se comunicar com outras aplicações. (Servidores de aplicação como: Jboss, WildFly, GlassFish)

# Pacote Jar

## Jar

- Cada classe em java com extensão .java se transforma em um .class;
- Um pacote.jar deve conter, entre outras coisas, os arquivos .class.
- Os arquivos .class são os referidos arquivos bytecode!
- Se você levar os seus arquivos .class para outro computador que tenha uma JVM instalada eles vão funcionar independente do S.O.

# Código Executável



# Código Interpretável

