**.JAVA- CRIAÇÃO DE PROJETOS**

**Baixei gradle, Maven,intelij**

##### Wrapper é uma forma de estabelecer controle de versões das ferramentas entre todos os membros da equipe.

**site do Spring initializar**

**Group-** Br.com.globalLEBS domio ao contraio

**Artifact** – é oprojeto que agnete ta trabalhando

**Name**-

**Description**-

**Packing**- jar

**Instalar jdk pelo Linux teminal**

##### - sudo add-apt-repositoty ppa:linuxprising/java

##### sudo apt update

##### III - sudo apt install oracle-java10-installer

##### java -version

**O QUE PRECISAMOS SABER SOBRE JAVA**

**O QUE É JAVA**? - É uma linguagem de programação e plataforma computacional lançada em 1995 pela **Sun Microsystems**, por um time comandado por **James Gosling.** Anos depois foi adquirida pela **Oracle**.

Diferente de outras linguagens de programação, que são **compiladas** para **código nativo**, o Java é compilado para um **bytecode** que é interpretado por uma **máquina virtua**l.

**O QUE É O COMPILADOR**? É um **programa** que a partir de um **código fonte**, cria um programa semanticamente equivalente, porem escrito em outra linguagem**, código objeto**. Um compilador traduz um programa de uma linguagem textual para uma linguagem máquina, especifica para um processador e sistema operacional.

O nome **compilador** é usado principalmente para os programas que **traduzem** o código fonte de uma **linguagem de programação de alto nível** (humano entende) para uma linguagem de programação de **baixo nível** (máquina entende) (por exemplo, Assembly ou código de máquina).

**O QUE É O BYTECODE?**

É o **código originado** da compilação de programas Java. Bytecode é o programa interpretado e executado pela máquina virtual Java **JVM**. **(.clas**)

**O QUE É JVM**? – **VM** é uma máquina virtual, é um software que simula uma máquina física e consegue executar vários programas, gerenciar processos, memoria e arquivos. Tudo isso faz parte de uma plataforma com memória, processador e outros recursos totalmente virtuais, sem dependência do hardware.

**JVM** é uma máquina virtual que executa programas Java, executando os ***bytecodes*** em linguagem de máquina para cada sistema operacional.

Em linguagens compiladas diretamente para um **SO** especifico, esse programa não irá executar em outro SO, havendo necessidade de compilar uma versão do software para cada SO.

Com o Java, compilamos para a JVM, o bytecode será executado pela máquina virtual, e não diretamente pelo SO, assim o software escrito em Java possui portabilidade para qualquer SO, porem cada JVM deve ser construída para um SO especifico.

**Código java (.java), Compilador (javac), Bytecode(.clas)**

**O QUE É JRE**? – ***Java Runtime Environment***, ou ambiente de **execução** do java, é composta pela **JVM, bibliotecas e APIs da linguagem Java e outros componentes** para suporte da plataforma Java. Ele representa a parte responsável pela execução do software Java. Responsável pra rodar o nosso software.

*Sudo apt-get install openjdk-8-jre*

**O QUE É JDK**?- Kit de desenvolvimento java, é um conjunto de **utilitários** que permitem criar software para a plataforma java. É composto pelo **compilador** java, **bibliotecas** da linguagem, **ferramentas e a JRE**.

*$ Sudo apt-get install openjdk-8-jdk*

**O QUE É JAVA SE**? – **Java Standard Edition** (SE), é a distribuição **mínima** da plataforma de desenvolvimento de aplicação java. Open JDK é a implementação de referência **opensource** da plataforma Java, Java SE, que ainda é mantida pela oracle.

**O QUE É JAVA EE**? Java **Enterprise Edition**, é uma **extensão** do Java **SE** que possui suporte a desenvolvimento de sistemas **corporativos**.

Além do mínimo da plataforma, o java EE possui diversas especificações de partes da infraestrutura de aplicações, como acesso a banco de dados, mensageria, serviços web, parser de arquivos e outras.

Servidores de aplicações java EE, sabem seguir essas especificações e implementar os recursos para usuários.

*Ex: JBoss (ReadHaf), Weblogic (Oracle), WebSphere (IBM) e*

*Glassfish = Implementacao de referencia OpenSource:* [*https://javaee.github.io/glassfish*](https://javaee.github.io/glassfish)

**O QUE É JAKARTA EE?** - Com a falta de investimento da Oracle no java, ela cedeu todo o código, implementações especificas do java EE para a **Eclipse Fundation**, mas como o nome java EE é uma marca registrada, foi escolhido o nome Jakarta EE.

Agora a evolução da especificações e padrões do java será feito sob o nome Jakarta EE, com compatibilidade com o java EE.

**JAVA- CARACTERISTICAS DA LINGUAGEM**

* Intelij 2019.1
* Java 11
* Gradle 5.3.1

**INTELIIJ**

**Grupo id**- nome do domínio do projeto: *one.Innovation.digital*

**Artefato**- nome do meu curso: *características-da-linguagem*

**Assuntos**

* **classes**
  + **main**,
  + **atributos**,
  + **métodos** (Retorno, parâmetros, assinatura) –todo método tem que ser **seguido de parênteses** e todo conteúdo entre nome do método e argumentos que ele recebe compõe a **assinatura do método**, não o retorno dele.
  + **construtores**
* **tipos**
  + **primitivos: Inteiros**
    - byte nullbyte=null;
    - nunca pode ser nulo e possuem valores default
    - 1 Byte= 8bits, aceita no máximo 127
    - recebe de -128 a 127
* **Char-**alfanumérico e **Short**
  + Ate 16 bits
* **Int e float**
  + 32 bits
* **Long e double**
  + 64 bits

**WRAPPERS**

* São ***objetos*** que representam os tipos primitivos
* **Auto –boxing**: **I*nteger, Byte, Float****...*
* Unboxing atribui um valor ao tipo primitivo

**Não primitivos**

* String, number, object, qualquer outros objetos

**Tipagem forte e estática**

* **Forte e estática**- *estática* quer dizer q os tipos de variáveis são verificados em **tempo de compilação É FORTE** porque ao atribuir um tipo, não posso mudar
* **Var- inferência de tipo**- consigo atribuir e criar as variáveis sem falar explicitamente o tipo, usando apenas o **var**.

**MODIFICADORES DE ACESSO** são sempre definidos em letra **minúsculas**.

* **PUBLIC**- pode ser acessado de qualquer entidade que possa visualizar a classe a que ela pertence.
* **PRIVATE**- os **métodos** e **atributos** da classe definidas como privados não podem ser acessados ou usados por nenhuma outra classe. Esses atributos e métodos também não podem ser visualizados pelas classes herdadas.
* **PROTECT**- torna o membro acessível as classes do ***mesmo pacote*** ou através de ***herança***, seus membros herdados ***não são acessíveis a outras classes*** fora do pacote em que foram declarados. É acessível também por *classes derivadas (extends*) em pacote diferentes.
* **Default**- A classe e/ou seus membros são acessíveis somente por classes do mesmo **pacote**, na sua declaração não é definido nenhum tipo de modificador, sendo este identificado pelo compilador.
* **Abstract-** é a **ideia** de uma classe, não tem corpo, conteúdo nenhum, esse modificador não é aplicado nas **variáveis**, apenas em **classes e métodos***. Uma classe abstrata não pode ser instanciada.* Se houver alguma declaração de um método como abstract a classe também deve ser marcada como abstract.
* **STATIC –** é usado para criação de uma **variável** que poderá ser acessada por todas as instâncias de objetos desta classe como uma variável comum, ou seja, a variável criada será a mesma em todas as instancias e quando seu conteúdo é modificado numa das instancias a modificação ocorre em todas as demais. E nas declarações de métodos ajudam no acesso direto a classe, portanto não é necessário instanciar um objeto para acessar o método.
* **Final-** quando é aplicado na classe, não permite **estender**, nos métodos impede que o mesmo seja sobrescrito (overriding) na subclasse, e nos valores de variáveis não pode ser alterado depois que já tenha sido atribuído um valor. Serve para atribuir valores constantes, ou seja, não pode ter seu conteúdo alterado depois de inicializado

**Super**-  é utilizada para fazer referência a métodos ou atributos da **super** classe.

**INTERFACES** obriga” a um determinado grupo de classes a ter métodos ou propriedades em **comum** para existir em um determinado contexto, contudo os métodos podem ser implementados em cada classe de uma maneira diferente. Pode-se dizer, a grosso modo, que uma interface é um contrato que quando assumido por uma classe deve ser implementado.

* **Métodos abstratos**- devem ser implementados para todos. Novos métodos quebram as implementações.
* **Métodos default**- são **herdados** a todos que implementam. Novos métodos não quebram as implementações.
* **Herança múltipla**\*\*\*

**ENUMS**

* Basicamente é dicionários de dados imutável.
* Não é permitido **criar novas instancias**
* O construtor é sempre declarado como **private**
* Por convenção, por serem objetos constantes e imutáveis (static final), os nomes são em **MAIUSCULOS**.

**CARACTERISTICAS DA LINGUAGEM II**

**Strings**- é uma classe que representa uma sequência de caracteres e está dentro do **Pacote java.lang**.

***Alguns métodos:***

***Trim*** tira os caracteres em branco, os espaços.

***toLowerCase()***- deixa tudo em minúsculo

***toUpperCase()***- deixa tudo em maiúsculo

***replace***- troca caracteres

***equals***- comparação

***equalsIgnoreCase***- ignora se está maiúsculo ou minúsculo

substring

contains

**String.format**- p imprimir concatenar

Estudar **verifica**

**Introducao a condicionais**

***IntStream forEach***- a partir do java 8 funcoes lambdas

**Convenções de nomes**

Nomes de classes- primeira letra maiúscula

Nomes de Métodos- minúsculos

Nomes de Variáveis – começa minúscula

**Code Style**

**Checkstyle Gradle** **Plugin-** verifica pequenas checks da qualidade de códigos.

**PMD Gradle Plugin**- faz analise de boas praticas, bugs

**Colocando os plug-ins no projeto**: pesquisar net

Abrir o build gradle-

**DEBUG DE CÓDIGO**

**TRABALHANDO COM DATAS**

**Classe** **Date**

**Java. util.Date-** está na jdk desde a versão 1.0.

**Construtores da classe:**

* **Date()** construtor padrão, vai alocar um objeto da classe Date e o **inicializará com o milissegundo mais próximo** do período da sua execução.
* **Date** (**long** date) diferente do construtor anterior, esse construtor espera que você passe os milissegundos com base padrão de tempo (**epoch**) que usa como referência 1 de janeiro de 1970 00:00:00. Vamos testar com base no
  + **System.curremtTimeMillis()** esse método estático vai retornar o milissegundo mais próximo de sua execução com base no **sistema operacional**

**O que é epoch?- epoch timestamp** é um padrão largamente aceito para representar uma data como um inteiro 32 bits a partir do início doo **Unix** **epoch.**

**Métodos uteis**



**Classe Instant-**

* surgiu jdk 1.8
* imutável e thread safe
* Modela um ponto instantâneo de uma linha do tempo
* Indicado para gravar marcações temporais em eventos da sua aplicação.

**Classe Calendar -Trabalhando com Datas**

Na jdk 1.1 foi observada a necessidade de facilitar alguns recursos que a class Date oferecia. Assim a classe **Calendar** foi criada. Com isso uma serie de métodos e construtores da classe Date foi depreciada. Por exemplo o construtor Date(int year, int month, int date).

**Calendar -**É uma classe abstrata que provê métodos para converter data entre um instante especifico. Possui alguns campos específicos para manipulação como MONTH, YEAR, HOUR, etc.

**Classe DateFormat – trabalhando com Datas**

Nesse ponto em que estamos existem, basicamente duas classes para formatação de datas. O **DateFormat** e o **SimpleDateFormat**. Ambos oferecem maneiras de formatar e parsear a saída das datas.

**SimpleDateFormat** traz uma grande facilidade que é definir um padrão de formatação para a saída de data que você deseja.

**Datas no Java 8+**

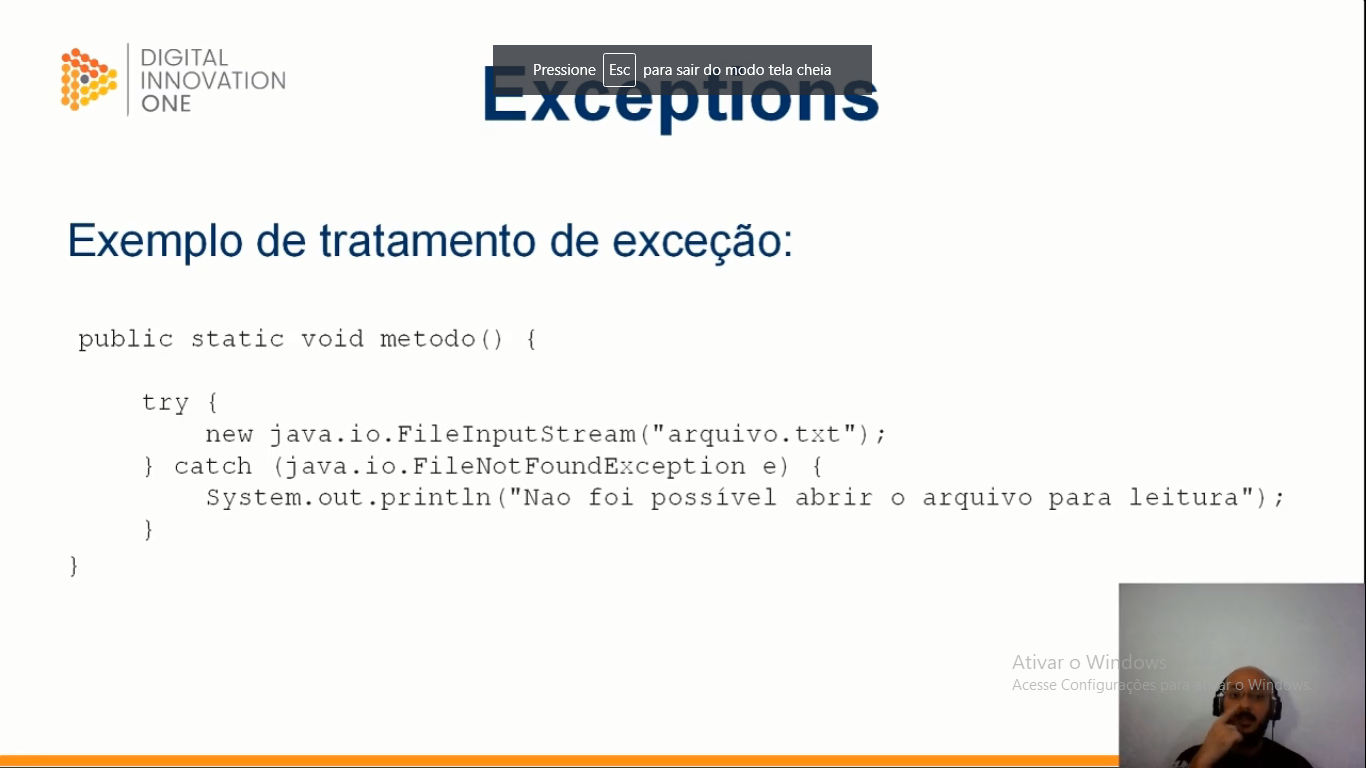
**LocalDate** é uma classe imutável para representar uma data. Seu formato padrão é **yyyy-MM-dd**

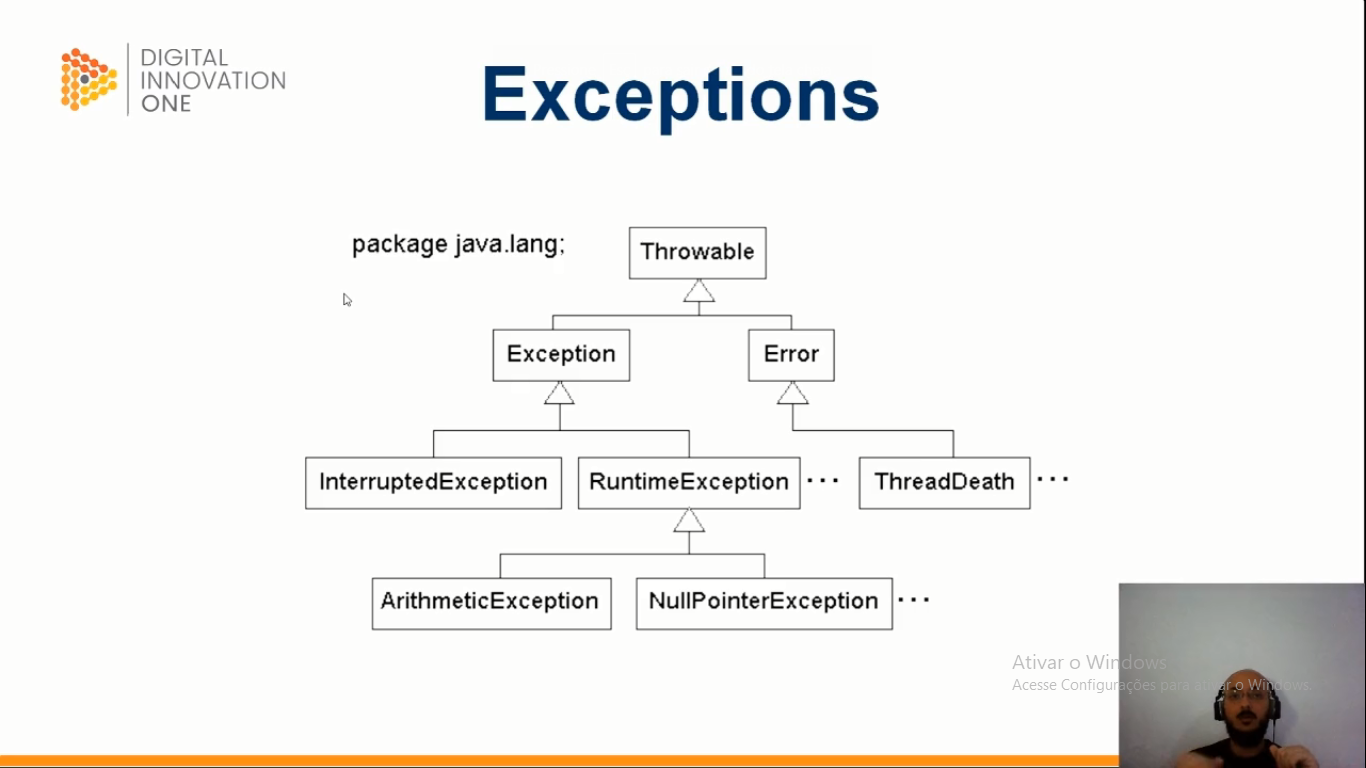
**LocalTime**- é uma classe imutável que representa um padrão de hora-minuto-segundo. Pode ser representado até o nível de nanosegundos. Ex: 12:22:10:12121223. Sua utilização é similar ao LocalDate

**LocalDateTime-** funciona como uma espécie de junção ente o localtime e o localdate. Também é uma classe imutável e você consegue trabalhar com dia e hora de uma so vez.

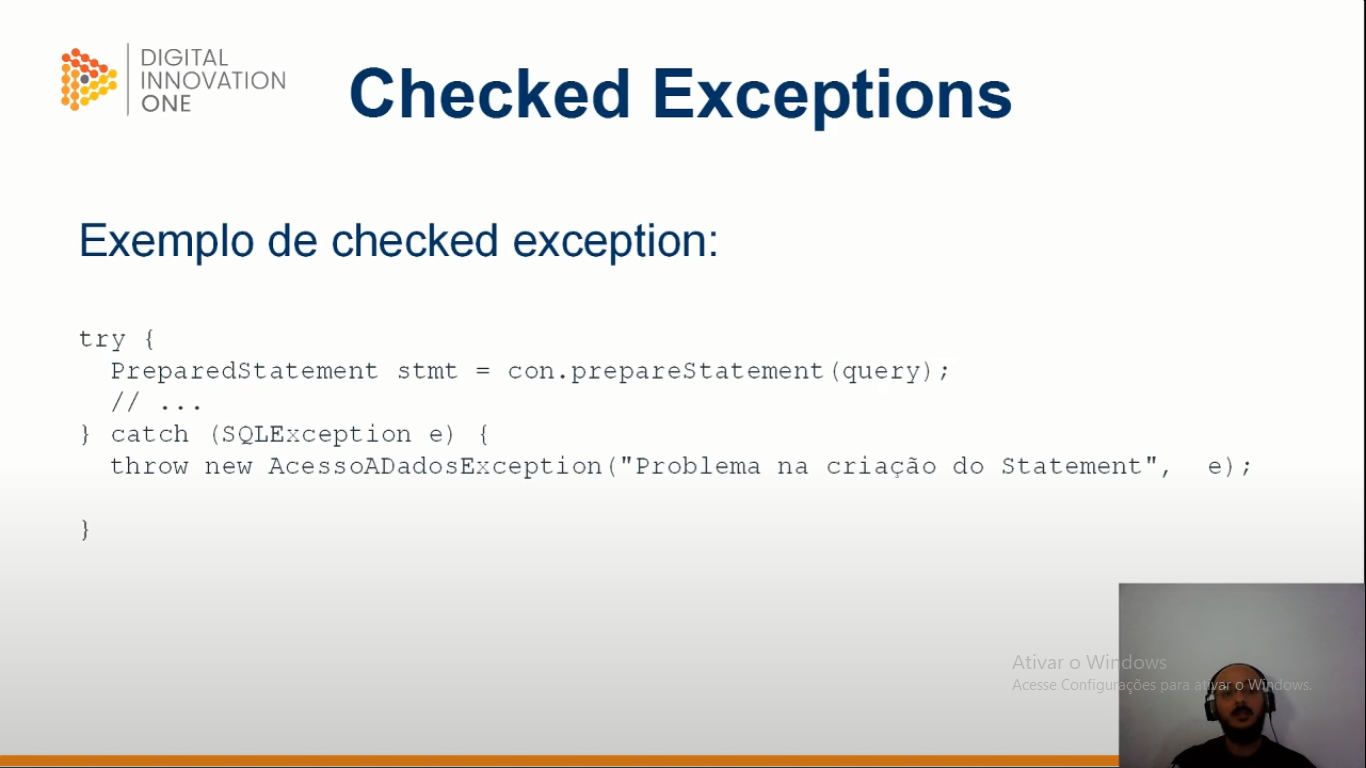
**TRATAMENTO DE EXCEÇÕES**

Exceções são todos os erros que ocorrem durante o processamento de um método que podem ser esperados ou não esperados. Exceptions são falhas que não devem ocorrer rotineiramente no fluxo de um sistema.

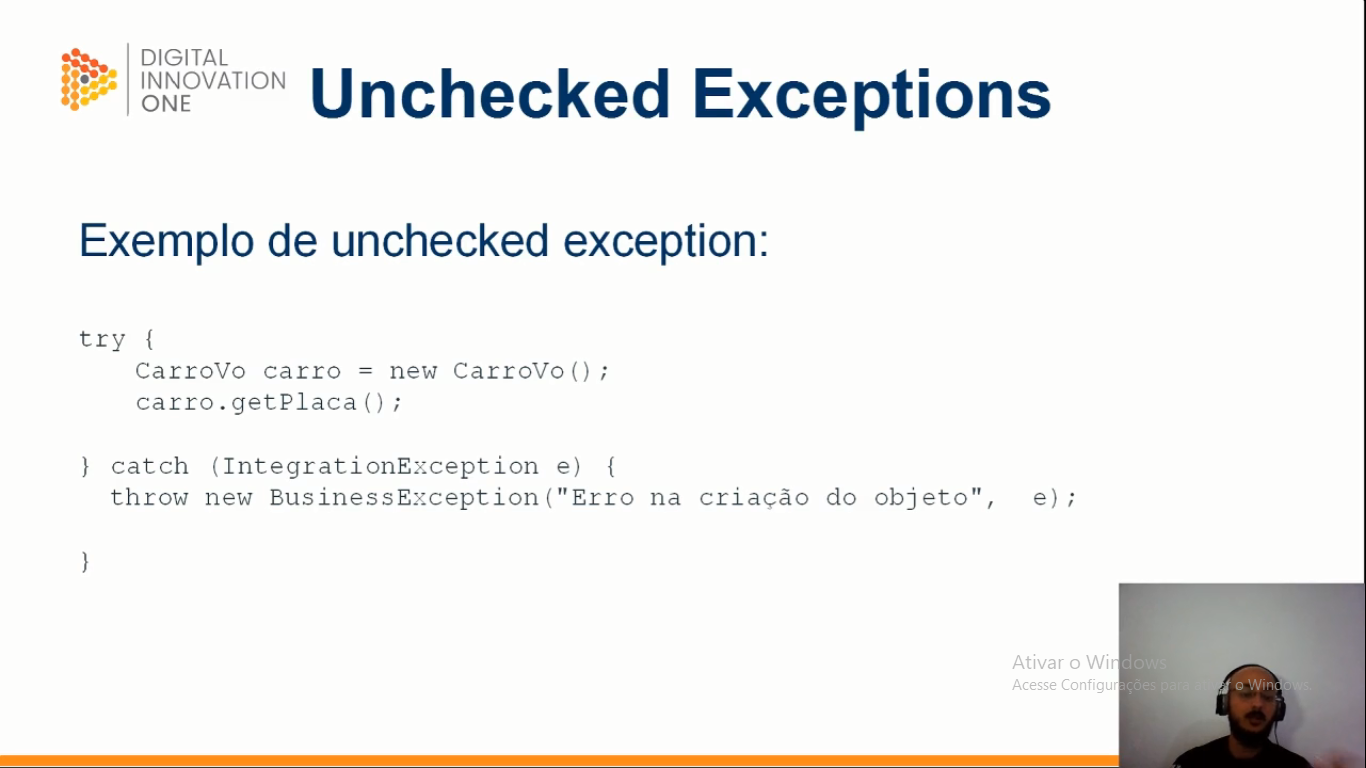




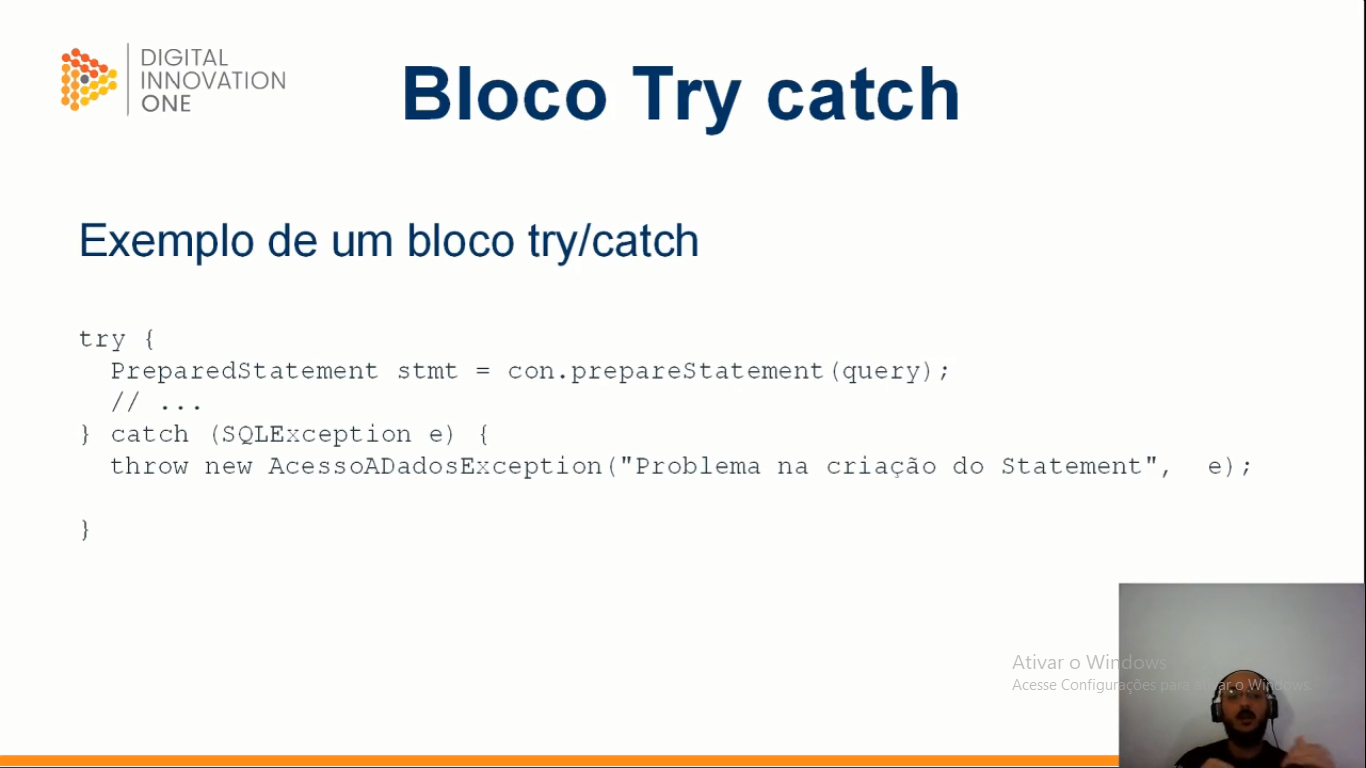
**Checked Exceptions**- São exceções esperadas, cujo fluxo ou método de um sistema foi preparado para receber. Um bom exemplo é uma exceção de negócio, onde se deseja informar um erro caso a exceção esperada ocorra.



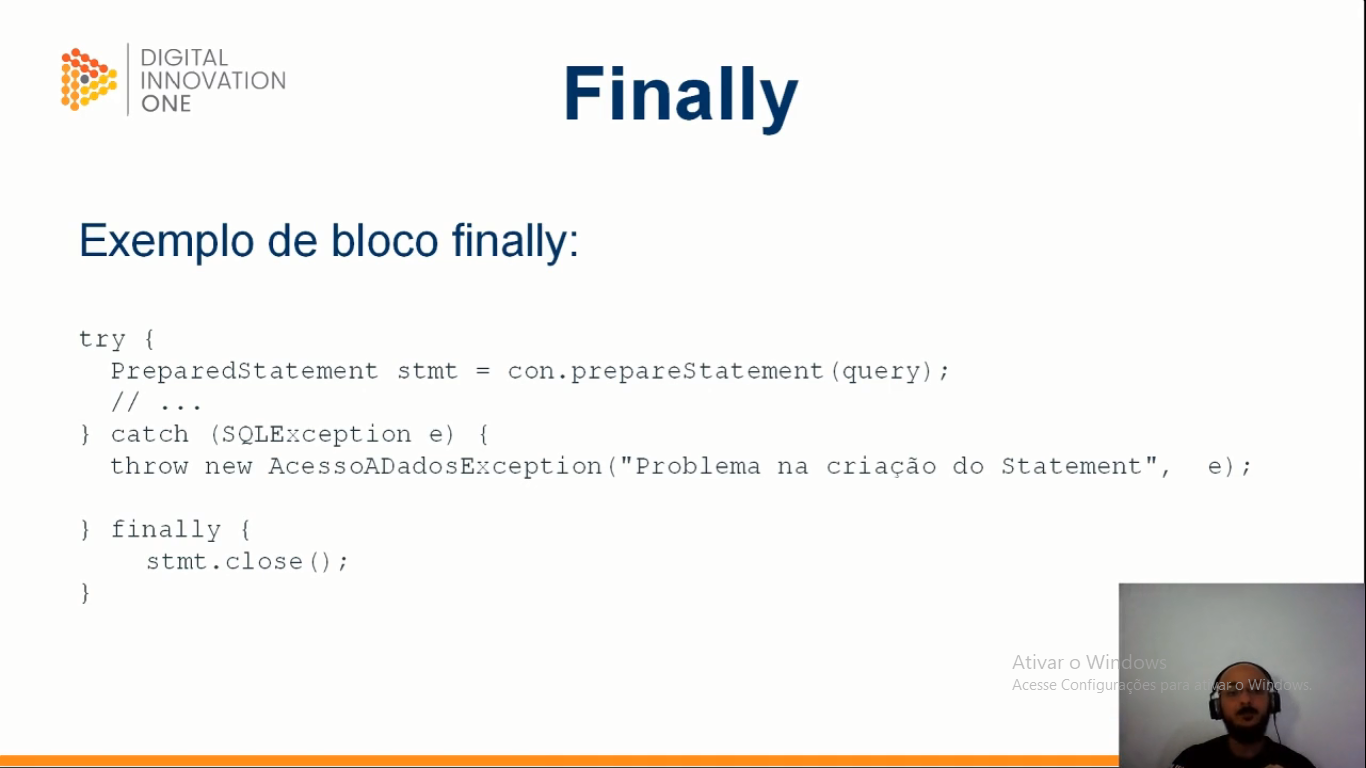
**Unchecked Exceptions**- São expressões não esperadas para o fluxo ou método de um sistema, um bom exemplo é a famosa NullPointException que ocorre quando se tenta acessar uma referencia de memoria vazia, ou recuperar uma instancia que não existe, dentre outros motivos.



**Bloco try catch-** sempre é utilizado quando no processo que será executado dentro de um método é esperado um erro, então cria-se um bloco “**protegido**” onde qualquer erro que ocorra dentro do trecho “try” é direcionado para o trecho “catch” e sofrera o devido tratamento de erro.



**Finally** é um bloco d código que pode ou não ser utilizado junto ao try catch, este trecho de código sempre sera executado independentemente se ocorrer erro ou não dentro do fluxo onde existe o try catch. Normalmente o finally é usado para liberar recursos ou para dar continuidade em um fluxo que deve ocorrer independente de erro.



**Throws**- é a assinatura do método que será retornado caso ocorra erro para o método que fez a chamada, dentro de um fluxo encadeado.

**Throw –** éusado para lançar a exceção desejada, juntamente com a mensagem de erro, para o método que fez a chamada.

