Programação Orientada a Objetos

Aula: Interfaces, Classes Abstratas e ENUM's.

Prof. Anderson Augusto Bosing

Classes Abstratas

Quando pensamos em um tipo de classe, supomos que os programas criam objetos desse tipo.
Às vezes é útil declarar as classes — chamadas classes abstratas — para as quais você nunca pretende criar objetos. Como elas só são utilizadas como superclasses em hierarquias de herança, são chamadas superclasses abstratas. Essas classes não podem ser utilizadas para instanciar objetos, porque, como veremos mais adiante, classes abstratas são incompletas. As subclasses devem declarar as "partes ausentes" para que se tornem classes "concretas", a partir das quais você pode instanciar objetos. Do contrário, essas subclasses também serão abstratas.

Classes Abstratas

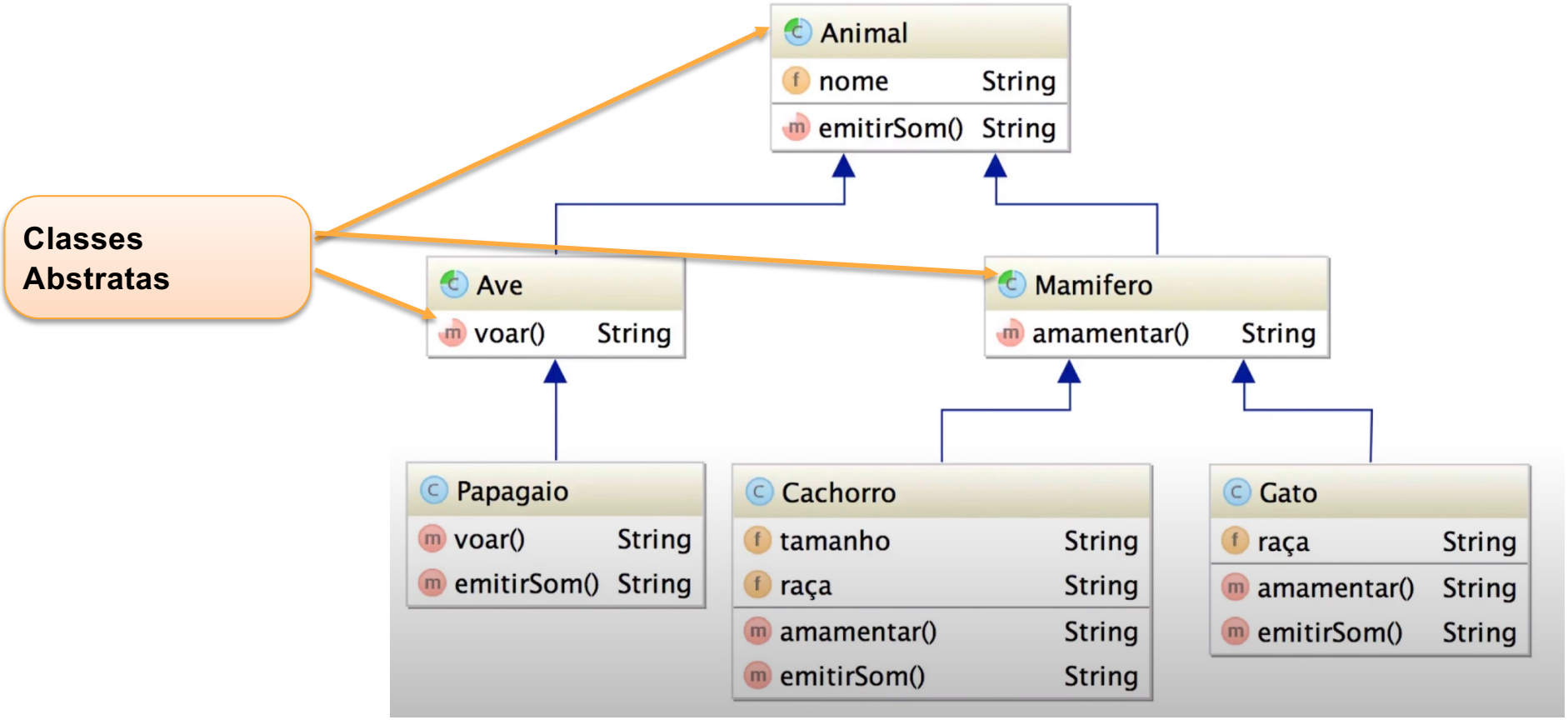
```java
/**
 *
 * @author andersonbosing
 */
public abstract class Funcionario {

    public abstract Double retornaBonificacao();

}
```

```java
 * @author andersonbosing
 */
public class Desenvolvedor extends Funcionario{

    @Override
    public Double retornaBonificacao() {
        return 1500.00;
    }


}
```

Classes Abstratas

Impossibilita a instanciação direta da classe.
Possibilita herança de código preservando comportamento.
- O que é genérico é herdado.
- O que é específico é implementado nas subclasses.
Adia especificidades de implementação para
Importância de Classes Abstratas Adia especificidades de implementação para fases posteriores de desenvolvimento.

Classes Abstratas



Classes Abstratas

Exemplos:

- Notas Entrada, Saída;

- Impostos;

- Pessoa, Aluno, Professor;

Interface

A interface é um recurso muito utilizado em Java, bem como na maioria das linguagens orientadas a objeto, para “obrigar” 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.

Interface

Uma interface não contém atributos, apenas assinaturas de métodos.
Não se pode instanciar objetos de interfaces, mas sim de classes que implementam as interfaces.

Interface

```
 */
public interface Banco {

    boolean saque(Conta conta, double valor);
    boolean deposito(Conta conta, double valor);
    void extrato(Conta conta);

}
```

```
 * @author andersonbosing
 */
public class BancoBrasil implements Banco {

    @Override
    public boolean saque(Conta conta, double valor) {
        throw new UnsupportedOperationException("Not supported yet."); //To chan
    }

    @Override
    public boolean deposito(Conta conta, double valor) {
        throw new UnsupportedOperationException("Not supported yet."); //To chan
    }

    @Override
    public void extrato(Conta conta) {
        throw new UnsupportedOperationException("Not supported yet."); //To chan
    }

}
```

Interface - Múltiplas Implementações

```
 * @author andersonbosing
 */
public class BancoBrasil implements Banco, Comparable {

    @Override
    public boolean saque(Conta conta, double valor) {
        throw new UnsupportedOperationException("Not supported yet."); //To chan
    }

    @Override
    public boolean deposito(Conta conta, double valor) {
        throw new UnsupportedOperationException("Not supported yet."); //To chan
    }

    @Override
    public void extrato(Conta conta) {
        throw new UnsupportedOperationException("Not supported yet."); //To chan
    }

    @Override
    public int compareTo(Object o) {
        throw new UnsupportedOperationException("Not supported yet."); //To chan
    }

}
```

https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html

Interface

Exemplos de Utilização:

- BasicDAO, FuncionarioDAO.

- BasicControllerRest.

- Pessoa implementando Comparable

Classes Enumeradas(Enum's)

São tipos de campos que consistem em um conjunto fixo de constantes, sendo como uma lista de valores pré-definidos. Na linguagem de programação Java, pode ser definido um tipo de enumeração usando a palavra chave enum.

Todos os tipos enums implicitamente estendem a classe java.lang.Enum, sendo que o Java não suporta herança múltipla, não podendo estender nenhuma outra classe.

Características dos Tipos Enum

Em relação às propriedades é preciso tomar os seguintes cuidados:
As instâncias dos tipos enum são criadas e nomeadas junto com a declaração da classe, sendo fixas e imutáveis (o valor é fixo).
Os construtores são obrigatoriamente privados muito embora não sejam necessário explicitar isso.
Não é permitido criar novas instâncias com a palavra chave new.
Acima de tudo são classes então podem ter construtores, atributos e métodos.
Seguindo a convenção, por serem objetos constantes e imutáveis, os nomes declarados recebem todas as letras em MAIÚSCULAS por convenção de código.

Enum - Exemplos

```java
public enum SexoEnum {

    MASCULINO,
    FEMININO,
    NAO_DECLARADO;

}
```

```java
public enum SexoEnum {

    MASCULINO("Masculino", "M"),
    FEMININO("Feminino", "F"),
    NAO_DECLARADO("Não Declarado", "NA");

    private String descricao;
    private String sigla;

    private SexoEnum(String descricao, String
        this.descricao = descricao;
        this.sigla = sigla;
    }

    public String getDescricao() {
        return descricao;
    }

    public String getSigla() {
        return sigla;
    }

    public void setSigla(String sigla) {
        this.sigla = sigla;
    }

}
```

Enum - Exemplos

Categorias de Filmes.
Sexo
Classes Medicamentos
Estado Civil
Genero
Dias da Semana

Chegou a sua vez..

- Lista de Exercícios de Herança, Enum, Interfaces com Java disponível no classroom.

- https://docs.google.com/document/d/1HSGzLcxrGI4JP8oWtqXXjRLAkwiVzVndZb41S5LikoM/edit?usp=sharing

Programação Orientada a Objetos

Aula: Tratamento de Exceções em Java.

O que são exceções?

O que são exceções?

Uma exceção é uma condição anormal que altera ou interrompe o fluxo de execução.

Indicação de um problema que ocorre durante a execução de um programa (DEITEL, 2005).

Podem ser causadas por diversas condições:

- **Erros sérios de hardware.**
- **Erros simples de programação.**
- **Erros de divisão por zero.**
- **Valores fora de faixa.**
- **Valores de variáveis.**
- **Erro na procura/abertura de um arquivo (entrada/saída).**
- **Falha na memória.**

Alguns problemas que geram exceções

- Pelo usuário:
  - Divisão por zero
  - Caracteres inválidos
- Pelo código:
  - Senha de acesso ao banco de dados errada
  - Tentativa de abrir um arquivo inexistente
  - Manipulação de variável com valor null (sem objeto)

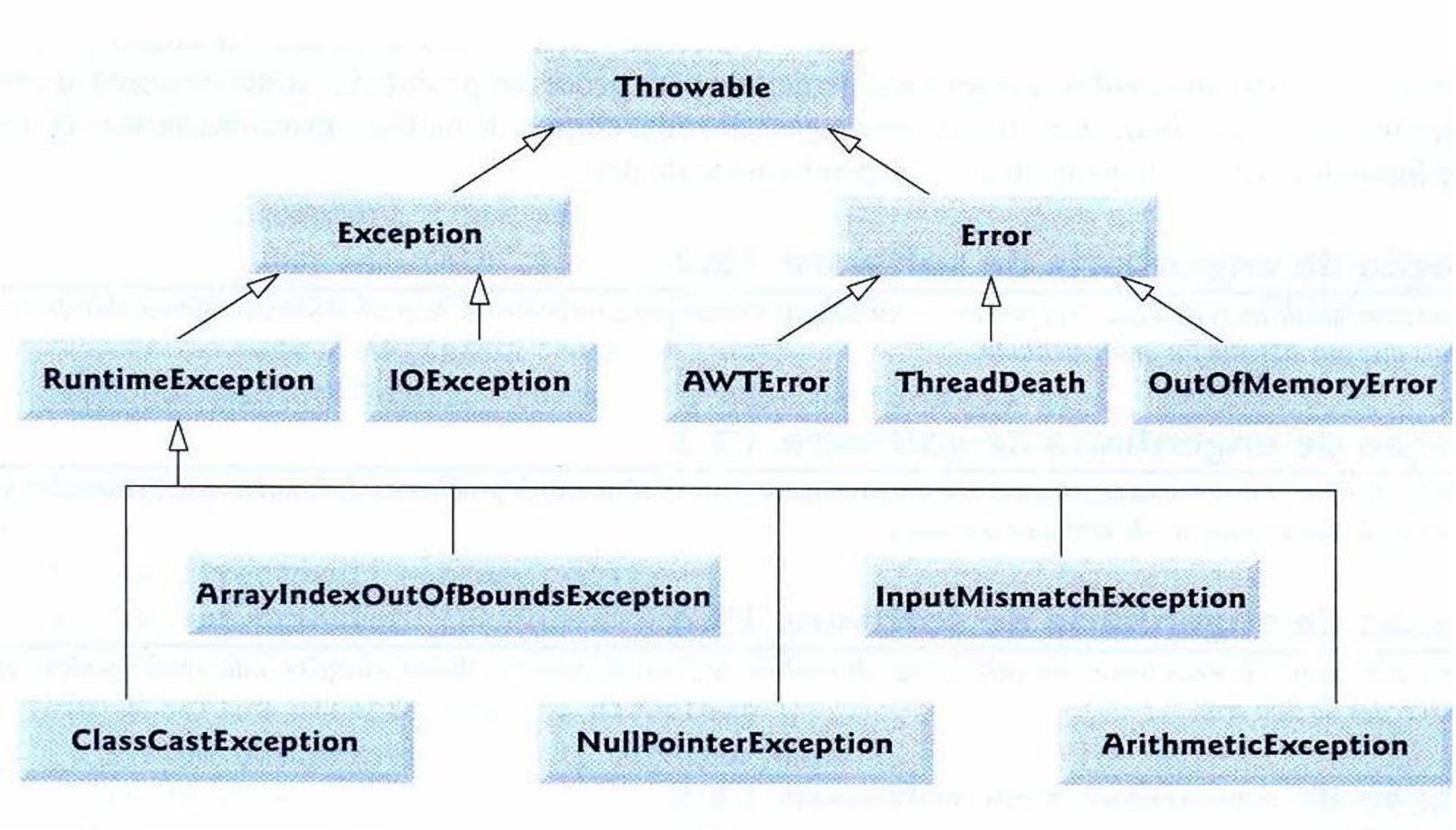
Tratamento de Exceções

- Mecanismo de identificar e tratar uma exceção.
- Permite que o programa continue executando.
- Favorece o desenvolvimento de aplicativos tolerante a falhas.

E uma exceção em Java é um ...?

OBJETO

Hierarquia de Exceções em Java



Classes de tratamento de exceções

Throwable: Classe base de todas as exceções.

Fornece os métodos:

- **getMessage() – retorna mensagem do erro.**
- **printStackTrace() – retorna a pilha de métodos chamados.**
- **toString - retorna uma string com a descrição da exception.**

Exception:

- **subclasse de Thworable.**
- **Por convenção, todas as classes de exceção são subclasses de Exception.**

Bloco try / catch

Como a exceção é lançada por toda a cadeia de classes do sistema, a qualquer momento é possível se "pegar" essa exceção e dar a ela o tratamento adequado.

Para se fazer este tratamento, é necessário pontuar que um determinado trecho de código que será observado e que uma possível exceção será tratada de uma determinada maneira:

O bloco try, é o trecho de código em que uma exceção é *esperada* e o bloco catch, em correspondência ao bloco try, prepara-se para *pegar* a exceção ocorrida e dar a ela o tratamento necessário. **Uma vez declarado um bloco try, a declaração do bloco catch torna-se obrigatória.**

```
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        try {
            /* Trecho de código no qual uma exceção pode acontecer.*/
        } catch (Exception ex) {
            /* Trecho de código no qual uma exceção do tipo "Exception" será tratada.*/
        }

    }

}
```

Exemplo em que as linhas abaixo da exception não serão executadas.

```java
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        Scanner s = new Scanner(System.in);

        try {
            System.out.print("Digite um valor inteiro..:");
            int numero1 = s.nextInt();
            System.out.print("Digite um valor inteiro..:");
            int numero2 = s.nextInt();

            System.out.println(numero1 + " + " + numero2 + " = " + (numero1 + numero2));
        } catch (Exception ex) {
            System.out.println("ERRO - Valor digitado nao e um numero inteiro!");
        }

    }
}
```

Palavra-chave throw

Também é possível que você próprio envie uma exceção em alguma situação especifica, como em uma situação de login em que o usuário digita incorretamente sua senha. Para realizarmos tal tarefa é necessária a utilização da palavra-chave throw da seguinte maneira:

throw new << Exceção desejada >>();

```
*/
public class Main {

    public static final String SENHASECRETA = "123456";

    public static void main(String[] args) {

        try {
            Scanner s = new Scanner(System.in);

            System.out.print("Digite a senha: ");

            String senha = s.nextLine();

            if (!senha.equals(SENHASECRETA)) {
                throw new Exception("Senha invalida!!!");
            }

            System.out.println("Senha correta!!!Bem vindo(a)!!!");

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }

    }

}
```

Bloco finnaly

A palavra-chave finally representa um trecho de código que será sempre executado, independentemente se uma exceção ocorrer.

```java
public class Main {

    public static final String SENHASECRETA = "123456";

    public static void main(String[] args) {

        try {
            Scanner s = new Scanner(System.in);

            System.out.print("Digite a senha: ");

            String senha = s.nextLine();

            if (!senha.equals(SENHASECRETA)) {
                throw new Exception("Senha invalida!!!");
            }

            System.out.println("Senha correta!!!Bem vindo(a)!!!");

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        } finally {
            System.out.println("Bloco Finally.");
        }

    }

}
```

Palavra-chave throws

Caso em algum método precise lançar uma exceção, mas você não deseja tratá-la, quer retorna-la para o objeto que fez a chamada ao método que lançou a exceção, basta utilizar a palavra chave throws no final da assinatura do método. Quando utilizamos o throws precisamos também informar qual ou quais exceções podem ser lançadas.

```java
public class Main {

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        try {
            Main et = new Main();

            System.out.print("Digite o valor do dividendo: ");
            double dividendo = s.nextDouble();

            System.out.print("Digite o valor do divisor: ");
            double divisor = s.nextDouble();

            double resultado = et.dividir(dividendo, divisor);

            System.out.println("O resultado da divisao eh: " + resultado);
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

    public double dividir(double dividendo, double divisor) throws Exception {
        if (divisor == 0) {
            throw new Exception("Nao e permitido fazer uma divisao por zero!");
        }

        return dividendo / divisor;
    }
}
```

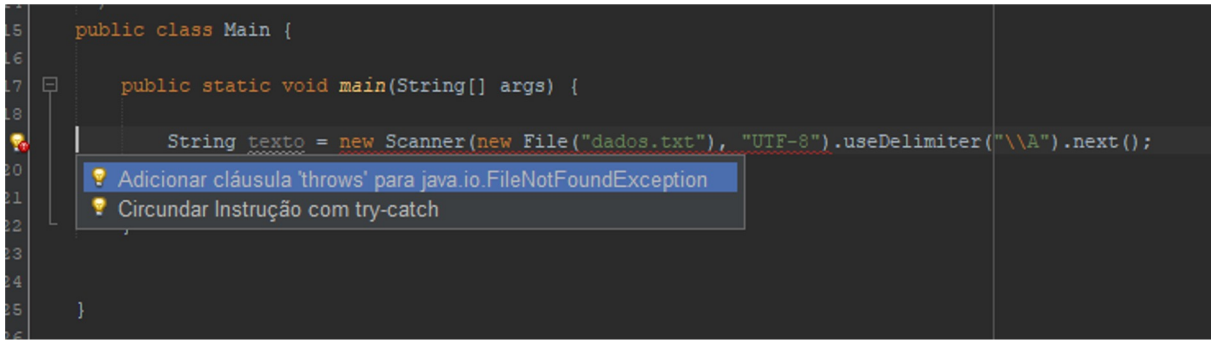
Tratamento de Exceções

Palavras utilizadas no tratamento de exceções:

- try
- catch
- finally
- throw
- throws

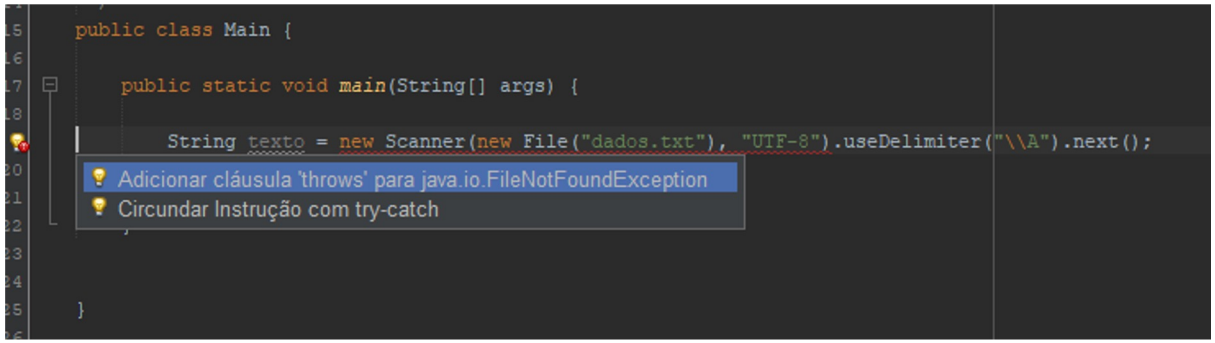
Checked Exceptions

As checked exceptions, são exceções já previstas pelo compilador. Usualmente são geradas pela palavra chave throw (que é discutido mais adiante). Como ela é prevista já em tempo de compilação, se faz necessária a utilização do bloco try / catch ou da palavra chave throws. A princípio, todas as classes filhas de Exception são do tipo checked, exceto pelas subclasses de java.lang.RuntimeException (exceção em tempo de execução).



Checked Exceptions

As checked exceptions, são exceções já previstas pelo compilador. Usualmente são geradas pela palavra chave throw (que é discutido mais adiante). Como ela é prevista já em tempo de compilação, se faz necessária a utilização do bloco try / catch ou da palavra chave throws. A princípio, todas as classes filhas de Exception são do tipo checked, exceto pelas subclasses de java.lang.RuntimeException (exceção em tempo de execução).



Unchecked Exceptions

Um dos fatores que tornam a RuntimeException e suas classes filhas tão específicas em relação as demais subclasses de Exception é que elas são exceções não diretamente previstas por acontecer em tempo de execução, ou seja, são unchecked exceptions. Por exemplo:

```
public class Main {

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.print("Digite um numero inteiro..: ");
        int numero = s.nextInt();

        System.out.println("Numero lido: " + numero);
    }

}
```

Este tipo de exceção, que acontece somente em tempo de execução, a princípio não era tratado e uma exceção do tipo java.util.InputMismatchException será gerada e nosso programa é encerrado. Agora iremos prever este erro, utilizando o bloco try / catch:

Unchecked Exceptions

```java
public class Main {

    public static void main(String[] args) {
        int numero = 0;
        boolean validado = false;
        while (!validado) {
            try {
                Scanner s = new Scanner(System.in);
                System.out.print("Digite um numero inteiro..: ");
                numero = s.nextInt();
                validado = true;
            } catch (InputMismatchException ex) {
                System.out.println("O valor inserido nao e um numero inteiro!");
            }
        }
        System.out.println("O valor lido foi: " + numero);
    }

}
```

Errors

Errors são um tipo especial de Exception que representam erros da JVM, tais como estouro de memória, entre outros. Para este tipo de erro normalmente não é feito tratamento, pois sempre quando um java.lang.Error ocorre a execução do programa é interrompida.

Tratando múltiplas Exceções

É possível se tratar múltiplos tipos de exceção dentro do mesmo bloco try / catch, para tal, basta declara-los um abaixo do outro, assim como segue:

```
public class Main {

    public static void main(String[] args) {
        try {
            /* Trecho de código no qual uma exceção pode acontecer. */
        } catch (InputMismatchException ex) {
            /* Trecho de código no qual uma exceção
         do tipo "InputMismatchException" será tratada. */
        } catch (RuntimeException ex) {
            /* Trecho de código no qual uma exceção
         do tipo "RuntimeException" será tratada. */
        } catch (Exception ex) {
            /* Trecho de código no qual uma exceção
         do tipo "Exception" será tratada. */
        }
    }

}
```

Vamos Praticar?

Lista de Exercícios no Classroom.

Criando sua exceção

Na linguagem Java podemos também criar nossas próprias exceções, normalmente fazemos isso para garantir que nossos métodos funcionem corretamente, dessa forma podemos lançar exceções com mensagens de fácil entendimento pelo usuários, ou que possa facilitar o entendimento do problema para quem estiver tentando chamar seu método possa tratar o erro.

No exemplo abaixo vamos criar uma exceção chamada ErroDivisao que será lançada quando ocorrer uma divisão incorreta, para isso precisamos criar esta classe filha de Exception.

```
*/
public class ErroDivisao extends Exception {

    public ErroDivisao() {
        super("Divisao invalida!!!");
    }
}
```

Criando sua exceção

```java
public class Main {

    public static void main(String[] args) {
        try {
            Scanner s = new Scanner(System.in);
            System.out.print("Digite o valor do dividendo: ");
            int numero1 = s.nextInt();

            System.out.print("Digite o valor do divisor: ");
            int numero2 = s.nextInt();

            Main teste = new Main();

            System.out.println("Resto: " + teste.restoDaDivisao(numero1, numero2));
        } catch (ErroDivisao ex) {
            System.out.println(ex.getMessage());
        }
    }

    public int restoDaDivisao(int dividendo, int divisor) throws ErroDivisao {
        if (divisor > dividendo) {
            throw new ErroDivisao();
        }

        return dividendo % divisor;
    }

}
```

Pesquisar sobre os conceitos abaixo:

Driver JDBC

DriverManager

Connection

ResultSet

PreparedStatement

Driver JDBC

Pode-se dizer que é uma API que reúne conjuntos de classes e interfaces escritas na linguagem Java na qual possibilita se conectar através de um driver específico do banco de dados desejado.

Com esse driver pode-se executar instruções SQL de qualquer tipo de banco de dados relacional.

Para fazer a comunicação entre a aplicação e o SGBDs é necessário possuir um driver para a conexão desejada. Geralmente, as empresas de SGBDs oferecem o driver de conexão que seguem a especificação JDBC para caso de algum desenvolvedor querer utilizar.

Pacote java.sql

Esse pacote oferece a biblioteca Java o acesso e processamento de dados em uma fonte de dado. As classes e interfaces mais importantes são:

DriverManager

Connection

ResultSet

PreparedStatement

DriverManager

Cada driver que se deseja a usar precisa ser registrado nessa classe, pois ela oferece métodos para gerenciar um driver JDBC.

Para a aplicação reconhecer a comunicação com o banco de dados escolhido, é preciso obter um arquivo com a extensão .jar que geralmente se consegue através dos sites das empresas que distribuem o SGBD. Esse arquivo tem o objetivo ajudar no carregamento do driver JDBC. Na prática de um desenvolvimento moderno essa biblioteca pode ser buscada e utilizada através do maven.

Connection

Representa uma conexão ao banco de dados.

Método close - Geralmente é inserido dentro do bloco finally para realizar sempre a sua execução, pois esse método serve para fechar e liberar imediatamente um objeto Connection.

Método prepareStatement - É usado para criar um objeto que representa a instrução SQL que será executada, sendo que é invocado através do objeto Connection.

PreparedStatement

Nesta interface que estende a interface base Statement são listados os métodos executeQuery e executeUpdate que são considerados os mais importantes referente a execução de uma query.

- executeQuery - Executa uma instrução SQL que retorna um único objeto ResultSet.
- executeUpdate - Executa uma instrução SQL referente a um INSERT, UPDATE e DELETE. Esse método retorna a quantidade de registros que são afetados pela execução do comando SQL.

ResultSet

Representa o conjunto de resultados de uma tabela no banco de dados. Esse objeto mantém o cursor apontando para a sua linha atual de dados, sendo que seu início fica posicionado na primeira linha.

Além disso, esse objeto fornece métodos getters referentes aos tipos de dados como: getInt, getString, getDouble, getFloat, getLong entre outros. Com esses métodos são possíveis recuperar valores usando, por exemplo, o nome da coluna ou número do índice.

```
conn = new DatabaseConnection().getConnection();

ps = conn.prepareStatement(FIND_ALL);
rs = ps.executeQuery();

while(rs.next()){

    Cor cor = new Cor();
    cor.setDescricao(rs.getString("descricao"));
    cor.setId(rs.getInt(1));

    listaResultado.add(cor);
}
```

Bibliografia Base

https://www.java.com/pt-BR/download/help/whatis_java.html. Acesso em 19/02/2022.

https://www.devmedia.com.br/java-historia-e-principais-conceitos/25178. Acesso em 19/02/2022.

DEITEL, Harvey; DEITEL, Paul. Java: Como programar. 10. ed. São Paulo: Pearson Education do Brasil, 2017.

Perguntas?

Conclusão