53 45 52 45 49 20 46 49 45 4c 20
41 4f 53 20 50 52 45 43 45 49 54
4f 53 20 44 41 20 48 4f 4e 52 41
20 45 20 44 41 20 43 49 c3 8a 4e
43 49 41 2c 20 50 52 4f 4d 4f 56
45 4e 44 4f 20 4f 20 55 53 4f 20
45 20 4f 20 44 45 53 45 4e 56 4f
4c 56 49 4d 45 4e 54 4f 20 44 41
20 49 4e 46 4f 52 4d c3 81 54 49
43 41 20 45 4d 20 42 45 4e 45 46
c3 8d 43 49 4f 20 44 4f 20 43 49
44 41 44 c3 83 4f 20 45 20 44 41
20 53 4f 43 49 45 44 41 44 45 2e

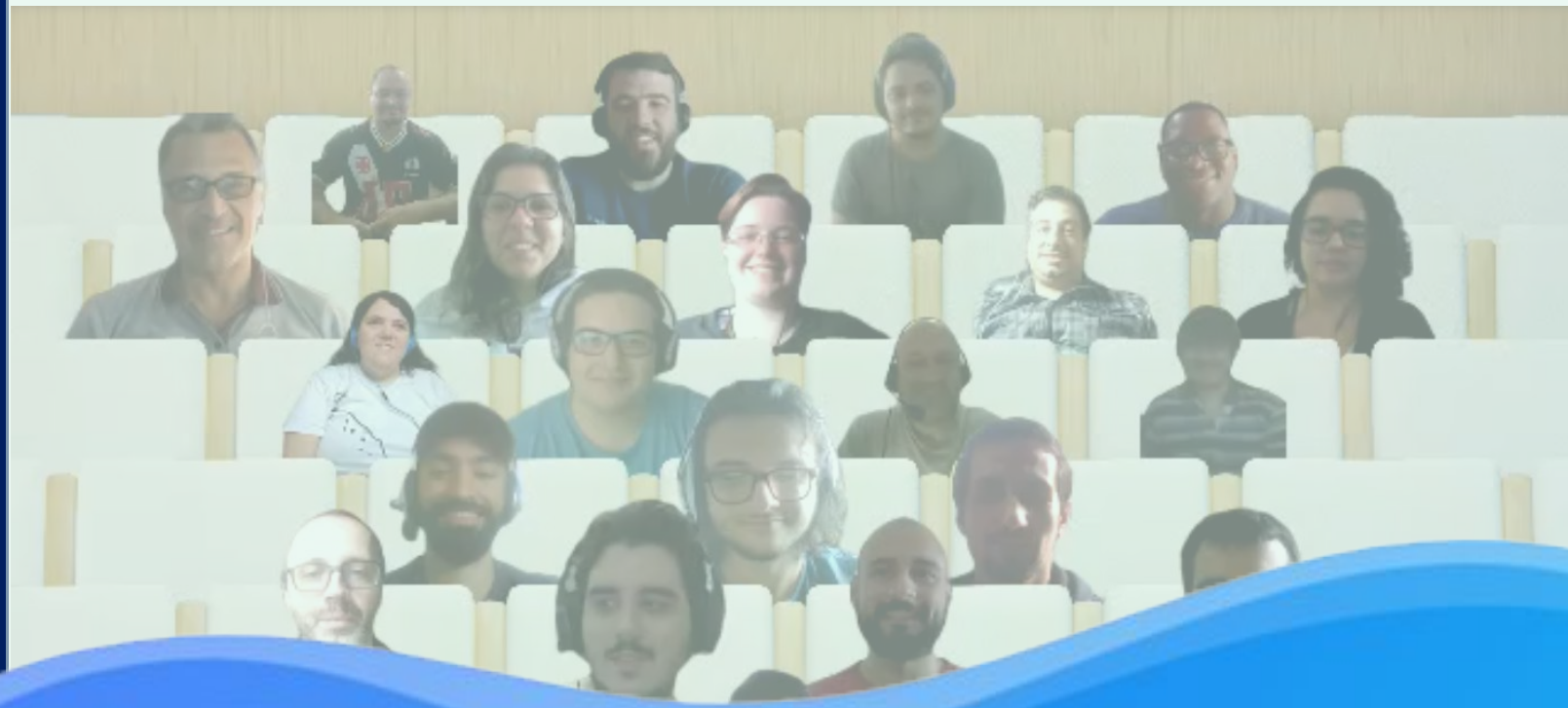
RESIDÊNCIA DE

SOFTWARE

CAPACITAR
TREINAR
EMPREGAR
TRANSFORMAR

Firjan SENAI

serratec
Parque Tecnológico Região Serrana



Genericcs, maps e data

Data: 11/09/2022

GENERICS

Generics

O T em <T> é um tipo que ele vai representar para uma determinada variável dentro de uma classe. Ele é usado na declaração de classes e de seus métodos. Ele é usado quando se quer criar uma classe onde suas variáveis são de um tipo que não é definido no momento em que ela é escrita, mas sim no momento que ela é usada, deixando a critério do usuário dessa classe qual será o tipo da variável no lugar do T.

Existem outras siglas padronizadas por convenção entre os desenvolvedores Java:

E - Elemento
K - Chave
N - Número
T - Tipo
V - Valor

```java
public class MeuGenerico<T> {
    private T var;

    public MeuGenerico(T var) {
        super();
        this.var = var;
    }

    public T getVar() {
        return var;
    }

    public void setVar(T var) {
        this.var = var;
    }
}
```

```java
public class Teste {

    public static void main(String[] args) {
        MeuGenerico<Integer> mg1 = new MeuGenerico<>(50);
        MeuGenerico<String> mg2 = new MeuGenerico<>("Olá");
        System.out.println(mg1.getVar());
        System.out.println(mg2.getVar());
    }
}
```

GENERICS

Já o **?**, no contexto de generics, basicamente serve como um wildcard, pois ele representa "qualquer tipo". Sua função é permitir o uso do polimorfismo junto com genéricos. Quando seguido pelo palavra reservada super, por exemplo <? super Number> ele aceita que qualquer objeto cujo supertipo é Number seja lido ou escrito a uma variável, pois é seguro tratar como Number qualquer subtipo dele.

```
public class Teste {

    public static void main(String[] args) {
        /*MeuGenerico<Integer> mg1 = new MeuGenerico<>(50);
        MeuGenerico<String> mg2 = new MeuGenerico<>("Olá");
        System.out.println(mg1.getVar());
        System.out.println(mg2.getVar());
        */
        MeuGenerico<? super Number> mg = new MeuGenerico<>();
        mg.setVar(new Integer(1));
        mg.setVar(50.5);
        mg.setVar(new BigInteger("1000"));
        //mg.setVar(new String("500"));
        System.out.println(mg.getVar());
    }
}
```

GENERICS

```java
public class Sorteio<T> {

    private List<T> lista;
    private Random random;

    public Sorteio() {
        lista = new ArrayList<>();
        random = new Random();
    }

    public void adicionar(T elemento) {
        lista.add(elemento);
    }

    public void deletar(T elemento) {
        lista.remove(elemento);
    }

    public T sorteio() {
        int pos = this.random.nextInt(lista.size());
        return lista.get(pos);
    }
}
```

```java
public class TesteSorteio {

    public static void main(String[] args) {
        Funcionario f1 = new Funcionario("Joao", "Engenheiro", 2000);
        Funcionario f2 = new Funcionario("Ana", "Analista", 3000);
        Funcionario f3 = new Funcionario("Julia", "Auxiliar", 4000);


        Sorteio<Funcionario> so = new Sorteio<>();
        so.adicionar(f1);
        so.adicionar(f2);
        so.adicionar(f3);

        Funcionario f = so.sorteio();

        System.out.println(f.getNome());

        Sorteio<Integer> so1 = new Sorteio<>();

        so1.adicionar(10);
        so1.adicionar(20);
        so1.adicionar(30);

        Integer i = so1.sorteio();
        System.out.println(i);
    }

}
```

GENERICS

```java
public class Calculo<T> {
    private T valor;

    public Calculo(T valor) {
        super();
        this.valor = valor;
    }

    @Override
    public String toString() {
        return "Calculo [valor=" + valor + "]";
    }

    public T getValor() {
        return valor;
    }

}
```

```java
public class ExemploGenerics {

    public static void main(String[] args) {
        Calculo<Integer> i = new Calculo<>(10);
        Calculo<Double> d = new Calculo<>(100.);
        Calculo<String> s = new Calculo<>("200");

        System.out.println(i);
        System.out.println(d);
        System.out.println(s);
    }

}
```

ORDENAÇÃO JAVA.LANG.COMPARABLE

Comparable é uma interface que quando as classes a implementam adquirem a propriedade de ordenação que permite que os objetos dessas classes sejam ordenados automaticamente. As classes **String** e **Date** implementam **Comparable** basta apenas usar o comando **Collections.sort(variável)** para ordenar a lista. A interface **Comparable** possui um único método: **int compareTo(Object o).** Este método deve retornar zero, se o objeto comparado for igual a este objeto, um número negativo, se este objeto for menor que o objeto dado, e um número positivo, se este objeto for maior que o objeto dado. Caso os objetos não possam ser comparados é lançada uma **ClassCastException.**

```
public class Pessoa implements Comparable<Pessoa> {
    private String nome;
    private String cpf;
    private int idade;

    public Pessoa(String nome, String cpf, int idade) {
        super();
        this.nome = nome;
        this.cpf = cpf;
        this.idade = idade;
    }

    @Override
    public String toString() {
        return "Pessoa [nome=" + nome + ", cpf=" + cpf + ", idade=" + idade + "]";
    }

    public String getNome() {
        return nome;
    }

    public String getCpf() {
        return cpf;
    }

    public int getIdade() {
        return idade;
    }

    @Override
    public int compareTo(Pessoa o) {
        return nome.compareTo(o.getNome());
    }

}
```

A classe String implementa **Comparable** implementando assim o método **compareTo**

ORDENAÇÃO JAVA.LANG.COMPARABLE

```java
import java.util.ArrayList;

public class TestePessoa {

    public static void main(String[] args) {
        Pessoa p1 = new Pessoa("João","90923409014",34);
        Pessoa p2 = new Pessoa("Mariana","10923409014",14);
        Pessoa p3 = new Pessoa("Helena","20923409014",55);
        List<Pessoa> listaPessoa = new ArrayList<>();
        listaPessoa.add(p1);
        listaPessoa.add(p2);
        listaPessoa.add(p3);

        Collections.sort(listaPessoa);
        System.out.println(listaPessoa);
    }

}
```

ORDENAÇÃO JAVA.LANG.COMPARABLE

Alterar o código abaixo na classe Pessoa

```
@Override
public int compareTo(Pessoa o) {
    if(idade < o.getIdade()) {
        return -1;
    }else if (idade > o.getIdade()){
        return 1;
    }else {
        return 0;
    }
}
```

Se a idade da Pessoa atual é menor do que da outraPessoa retormamos -1 ou qualquer inteiro negativo, se for maior retornamos 1 ou qualquer inteiro positivo e se for igual então retornamos 0.

O método pode funcionar somente com esta linha.

```
return idade - o.getIdade();
```

Se trocarmos o atributo de int para Integer que implementa Comparable podemos fazer desta forma

```
@Override
public int compareTo(Pessoa o) {
    return idade.compareTo(o.getIdade());
}
```

MAPAS - JAVA.UTIL.MAP

Um mapa é composto por um conjunto de associações entre um objeto chave a um objeto valor. É equivalente ao conceito de dicionário, usado em várias linguagens. Não permite repetições da chave sendo indexados pela chave.

Suas principais implementações são:

- HashMap - não ordenado consequetemente mais rápido
- TreeMap - ordenado.
- LinkedHashMap - Os elementos ficam na ordem que são adiconados.

MAPAS - JAVA.UTIL.MAP

Exemplo:

```java
public class Exemplo1Map {
    public static void main(String[] args) {
        HashMap<String, String> mapaEstados = new HashMap<>();
        mapaEstados.put("AC", "ACRE");
        mapaEstados.put("AL", "ALAGOAS");
        mapaEstados.put("RJ", "RIO DE JANEIRO");
        System.out.println(mapaEstados.keySet());
        //Retorna chave
        for (String string : mapaEstados.keySet()) {
            System.out.println(string + "");
        }
        System.out.println("-------------------------");
        //Retorna valor
        for (String string : mapaEstados.values()) {
            System.out.println(string + "");
        }
        System.out.println("-------------------------");
        //Retorna chave e valor
        for (Map.Entry<String, String> mapa : mapaEstados.entrySet()) {
            System.out.println(mapa);
            System.out.println(mapa.getKey() + "-" + mapa.getValue());
        }
    }
}
```

Percorrendo Elementos obtendo a chave

Percorrendo Elementos obtendo o valor

Percorrendo Elementos obtendo a chave e o valor

MAPAS - JAVA.UTIL.MAP

A sintaxe deve obedecer a lugares apontados da chave e valor, pois cada chave leva somente um elemento.
Map<E> mapa = new Type();

Sintaxe:
E - é o objeto declarado, podendo ser classes Wrappers ou tipo de coleção.
Type - é o tipo de objeto da coleção a ser usado.

Classe HashMap
Os elementos não são ordenados. É rápida na busca/inserção de dados. Permite inserir valores e chaves nulas;

```
public class TesteMap {
    public static void main(String[] args) {
        Map<Integer, String> mapaNomes = new HashMap<Integer, String>();
        mapaNomes.put(1, "João");
        mapaNomes.put(2, "Maria");
        mapaNomes.put(3, "Gerson");

        mapaNomes.replace(2, "Ana");  //Modificando conteúdo
        mapaNomes.remove(3);

        System.out.println(mapaNomes);

        System.out.println(mapaNomes.get(2));
        System.out.println(mapaNomes.keySet());
        System.out.println(mapaNomes.values());
        System.out.println(mapaNomes.entrySet());

        for(int i=1; i< mapaNomes.size(); i++) {
            System.out.println(mapaNomes.get(i));
        }

    }
}
```

MAPAS - JAVA.UTIL.TREEMAP

Diferença entre TreeMap e HashMap em Java

TreeMap	HashMap
Mantém ordem crescente	Não mantém ordem e também não ordena
Não pode conter chave nula	Pode conter uma chave nula
Não permite valores nulos	Permite valores nulos
A classificação é mais lenta	A manipulação dos elementos é mais rápida

```
TreeMap<String, String> treeMaps = new TreeMap<>(mapaEstados);
System.out.println(treeMaps);
System.out.println(treeMaps.descendingMap());
System.out.println(treeMaps.descendingKeySet());
```

MAPAS - JAVA.UTIL.LINKEDHASHMAP

No LinkedHashMap a ordem como os elementos são inseridos é mantida. O HashMap oferece a vantagem de inserção, pesquisa e exclusão rápidas, mas nunca mantém o controle e a ordem de inserção que o LinkedHashMap fornece, onde os elementos podem ser acessados em sua ordem de inserção.

Um LinkedHashMap contém valores baseados na chave. Ele implementa a interface Map e estende a classe HashMap .
Ele contém apenas elementos únicos. Ele pode ter uma chave nula e vários valores nulos.
É o mesmo que HashMap com um recurso adicional que mantém a ordem de inserção. Por exemplo, quando executamos o código com um HashMap, obtemos uma ordem diferente de elementos.

```
public class ExemploLinkedHashMap {

    public static void main(String[] args) {
        LinkedHashMap<Integer, String> cursos = new LinkedHashMap<>();
        cursos.put(1, "Orientação Objetos");
        cursos.put(2, "Java Web");
        cursos.put(3, "API REST");
        cursos.put(4, "HTML");

        for (Map.Entry<Integer, String> curso : cursos.entrySet()) {
            System.out.println(curso);
        }

    }

}
```

EXERCÍCIOS

Criar uma classe para inserir em um ´mapa de marcas e modelos de alguns veículos e percorrer para imprimir a chave o valor.

```
public class TesteVeiculo {
    public static void main(String[] args) {
        Map<String, String> carros = new HashMap<String, String>();
        carros.put("VW", "Gol");
        carros.put("Fiat", "Siena");
        carros.put("Ford", "Fiesta");
        carros.put("Renault", "Sandero");

        // Obtendo a chave - Marcas dos carros
        for (String s : carros.keySet()) {
            System.out.println(s);
        }

        // Obtendo a chave e o valor - Marca e modelo dos carros
        for (Map.Entry<String, String> entrada : carros.entrySet()) {
            System.out.println(entrada);
        }
    }
}
```

TRABALHANDO COM DATAS NO JAVA

Classe Date

A data representa o tempo, um tempo é composto por ano, mês, dia atual, minuto atual, entre outros atributos e métodos que essa classe possui. Hoje a maioria dos métodos da classe Date estão classificados como deprecated, métodos que não são mais utilizados. A classe Date foi substituída pela Calendar.

```
public class Datas {

    public static void main(String[] args) {

            Date dataDeHoje = new Date();

            System.out.println("Data de Hoje:" + dataDeHoje);
            System.out.println("Milisegundos desde 1 janeiro de 1970:" + dataDeHoje.getTime());
            System.out.println("Dia de Hoje" + dataDeHoje.getDate());

            SimpleDateFormat formato = new SimpleDateFormat("dd/MM/yyyy");
            String dataFormatada = formato.format(dataDeHoje);
            System.out.println("Data Formatada:" + dataFormatada);

    }

}
```

TRABALHANDO COM DATAS NO JAVA

Classe Calendar

É uma classe abstrata que não pode ser instanciada, portanto para obter um calendário é necessário usar o método estático.
A classe produz valores de todos os campos de calendário necessários para implementar a formatação de data e hora, para uma determinada língua e estilo de calendário.

```
public class Calendars {

    public static void main(String[] args) {
        Calendar hoje = Calendar.getInstance();
        System.out.println(hoje);

        int ano = hoje.get(Calendar.YEAR);
        int mes = hoje.get(Calendar.MONTH);
        int dia = hoje.get(Calendar.DAY_OF_MONTH);
        int hora = hoje.get(Calendar.HOUR_OF_DAY);
        int minutos = hoje.get(Calendar.MINUTE);
        int segundos = hoje.get(Calendar.SECOND);

        System.out.println(dia + "-" + mes + "-" + ano);
        System.out.println(hora);

        //mês em Calendar começa com 0
        System.out.printf("Hoje é: %02d/%02d/%d",dia,mes+1,ano);
        System.out.printf("\nHora: %d:%d:%02d",hora,minutos,segundos);
    }

}
```

TRABALHANDO COM DATAS NO JAVA

Classe LocalDate

A manipulação de datas usando **Date** e **Calendar** para criação de filtros, cálculos e conversões eram trabalhosas nessas classes.
E também escrevíamos muito código. A partir do Java 8 foi disponibilizada uma nova API de data e hora. O **LocalDate** foi introduzido para facilitar o trabalha com datas. Ele fica no pacote **java.time**

```
public class TesteLocalDate {

    public static void main(String[] args) {
        LocalDate hoje = LocalDate.now();
        System.out.println(hoje);

        System.out.println(LocalDate.of(2020, 8, 10));

        System.out.println(LocalDate.parse("2020-08-10"));

        //ADICIONANDO 30 DIAS
        System.out.println(hoje.plusDays(30));
        //8 DIAS ATRÁS
        System.out.println(hoje.minusDays(8));
        //DOIS MESES ATRÁS
        System.out.println(hoje.minus(2, ChronoUnit.MONTHS));
        System.out.println(hoje.getDayOfWeek() + " " + hoje.getDayOfMonth() +" " + hoje.getDayOfYear());
        //SE É ANO BISEXTO
        System.out.println(hoje.isLeapYear());

    }

}
```

TRABALHANDO COM DATAS NO JAVA

Trabalhando com comparações de Datas

Frequentemente quando trabalhamos com datas necessitamos fazer operações de comparação, diferença entre datas, etc. Para isso, essa nova API veio com outras facilidade para que possamos fazer essas operações de maneira mais simples.

- isAfter
- isBefore
- isEquals
- LocalDate.now()
- plusDays()
- minusYears()

Uma classe que também nos ajuda nessa comparação é a **Period**. Ela representa uma quantidade de tempo em anos, meses e dias

```
Period periodo = Period.between(dataInicio, dataFim);
periodo.getYears()
```

← Cálculo de anos entre duas datas

EXERCÍCIO

1. Escreva um programa que escreva na console o dia de hoje, o dia da semana, mês e ano atual.
2. Alem disso, exiba quanto tempo se passou desde sua data de nascimento.

```
public class ExemploLocalDate {

    public static void main(String[] args) {
        LocalDate dataHoje = LocalDate.now();
        LocalDate dataNascimento = LocalDate.of(1975, 6, 11);
        System.out.println("Dia da semana:"+dataNascimento.getDayOfWeek().name());
        System.out.println("Dia da semana:"+dataNascimento.getDayOfWeek().ordinal());
        System.out.println("Mês:"+ dataNascimento.getMonthValue());
        System.out.println("Mês:"+ dataNascimento.getMonth().name());
        System.out.println("Ano:"+ dataNascimento.getYear());

        Period period = Period.between(dataNascimento, dataHoje);
        System.out.println("Passaram:"+ period.getYears()+ " anos");
        System.out.println(period.getMonths()+ " meses");
        System.out.println(period.getDays()+ " dias");
    }
}
```

TRABALHANDO COM HORAS NO JAVA

Classe LocalTime

Função para manipulação de horas.

```
public class TesteLocalTime {

    public static void main(String[] args) {
        LocalTime hora = LocalTime.now();
        System.out.println(hora);

        System.out.println(LocalTime.of(20, 10));
        System.out.println(LocalTime.parse("20:10"));

        System.out.println("Hora Atual + 60 minutos: "+hora.plusMinutes(60));
        System.out.println("Hora Atual - 10 minutos: " + hora.minusMinutes(10));


    }

}
```

TRABALHANDO COM DATAS E HORAS NO JAVA

Classe LocalDateTime

```
public class TesteLocalDateTime {

    public static void main(String[] args) {
        LocalDateTime dataHora= LocalDateTime.now();
        System.out.println(dataHora);

        System.out.println(dataHora.plusDays(10));

        System.out.println(LocalDateTime.of(2020,8,10,13,00,00));

        //Exibe fuso padrão do sistema
        ZoneId fuso = ZoneId.systemDefault();
        System.out.println(fuso);

        //Atribui um fuso
        fuso = ZoneId.of("America/Sao_Paulo");

        //Exibe todos os fusos disponíveis
        Set<String>fusos = ZoneId.getAvailableZoneIds();
        for(String f: fusos) {
            System.out.println(f);
        }
    }
```

TRABALHANDO COM DATAS E HORAS NO JAVA

Duration de tempo entre dois objetos LocalDateTime

Podemos utilizar a classe **Duration** para nos auxiliar no calculo da duração entre dois objetos **LocalDateTime**

```
public class ExemploDuration {

    public static void main(String[] args) {

        LocalDateTime primeiroPeriodo = LocalDateTime.of(2022, Month.JANUARY, 10, 16, 30, 00);
        LocalDateTime segundoPeriodo = LocalDateTime.of(2022, Month.JANUARY, 10, 20, 00, 00);

        Duration d1 = Duration.between(primeiroPeriodo, segundoPeriodo);
        long dias = d1.toDays();
        Duration d2 = d1.minus(dias, ChronoUnit.DAYS);
        long horas = d2.toHours();
        Duration d3 = d2.minus(horas, ChronoUnit.HOURS);
        long minutos = d3.toMinutes();
        Duration d4 = d3.minus(minutos, ChronoUnit.MINUTES);
        long segundos = d4.getSeconds();
        Duration d5 = d4.minus(segundos, ChronoUnit.SECONDS);
        long nanos = d5.toNanos();
        Duration d6 = d5.minus(nanos, ChronoUnit.NANOS);

        System.out.println("Total: " + dias + " dias, " + horas + " horas, " + minutos + " minutos, " + segundos
                + " segundos, " + nanos + " ns.");
        System.out.println("Resultado: " + d1.toString());
    }

}
```

TRABALHANDO COM DATAS E HORAS NO JAVA

Formatando datas e horas

A formatação de data para diferentes padrões também ficou um pouco mais simples nessa nova versão, para isso, agora é criado um formatador de dado com a classe **DateTimeFormatter** e a própria classe **LocalDate** tem um método **format** que retorna uma String com a data formatada no padrão passado como parâmetro.

```
public class MainLocalDateFormat {
    public static void main(String[] args) {
        LocalDate hoje = LocalDate.now();
        DateTimeFormatter formatadorBarra = DateTimeFormatter.ofPattern("dd/MM/yyyy");
        DateTimeFormatter formatadorTraco = DateTimeFormatter.ofPattern("dd-MM-yyyy");

        System.out.println("Data com /: " + hoje.format(formatadorBarra));
        System.out.println("Data com -: " + hoje.format(formatadorTraco));
    }
}
```

EXERCÍCIO – REVISANDO CONCEITOS DE AULA ANTERIORES - EXCEPTION

Classe Scanner

Facilita a entrada de dados no Java, surgiu a partir do Java 5 com o objetivo de facilitar a entrada de dados no modo Console.
Uma das características mais interessante da classe Scanner é a possibilidade de obter o valor digitado diretamente no formato do tipo primitivo que o usuário digitar.
Para isso basta utilizarmos os métodos next do tipo primitivo no formato nextTipoDado() conforme exemplo abaixo:

```
Scanner scanner = new Scanner(System.in);
int numeroInteiro = scanner.nextInt();
byte numeroByte = scanner.nextByte();
long numeroLong = scanner.nextLong();
boolean booleano = scanner.nextBoolean();
float numeroFloat = scanner.nextFloat();
double numeroDouble = scanner.nextDouble();
```

Exemplo:

```
public class Console {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.println("Qual o seu nome:");
        String nome= sc.next();
        System.out.println("Bom dia !!" + nome );
    }

}
```

EXERCÍCIO – REVISANDO CONCEITOS DE AULA ANTERIORES - EXCEPTION

Exercício Exceções

Criar um programa que leia um número inteiro e trate a exceção caso uma letra seja digitada. Tratar a exceção do tipo **InputMismatchException**

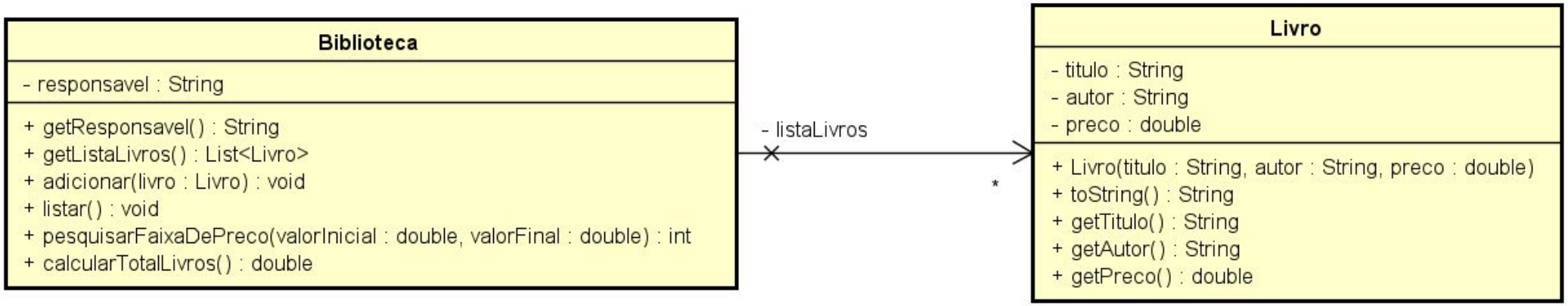
```
public class Console {

    public static void main(String[] args) {
        try {
            Scanner sc = new Scanner(System.in);
            System.out.println("Digite um número:");
            int numero = sc.nextInt();

        } catch (InputMismatchException e) {
            System.out.println("Erro !! Digite um número inteiro !!");
        }
    }
}
```

EXERCÍCIO – REVISANDO CONCEITOS DE AULA ANTERIORES

Criar a classe conforme diagrama abaixo:
A classe Biblioteca deverá ter um método para adicionar um novo livro, listar todos os livros, pesquisar por faixa de preço, calcular o total em dinheiro de livros.
Criar uma classe com o método main para fazer os testes das operações no console.



EXERCÍCIOS

- Implemente uma classe em Java com as funcionalidades de uma agenda telefônica, associando um nome a um número telefônico.
- Crie um programa em Java para testar a classe AgendaTelefonica. Teste a classe com pelo menos 5 contatos diferentes na agenda de telefones.
- A classe deve ser de acordo com o diagrama abaixo: