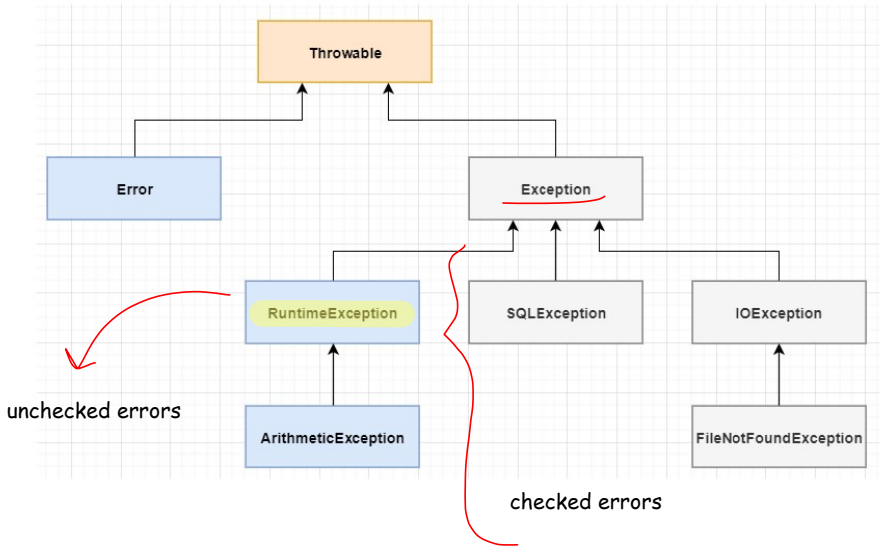
O Erro:

- Deve ser uma exceção e não o comportamento esperado
- Deve ser manuseado para criar aplicativos confiáveis
- Pode ocorrer como resultado de bugs de aplicativo
- Pode ocorrer devido a fatores fora do controle do aplicativo
    - Bancos de dados tornando-se inacessíveis
    - Os discos rígidos falham



Atenção: Captura de erros

Os erros lançados pelos métodos devem ser capturados de maneira específica nos blocos **catch** para facilitar a identificação e correção. Capturar erros de forma genérica e com mensagens genéricas irá dificultar a manutenção do programa.

try-with-resources

classes que implementam AutoCloseable

```
1:  public class TesteTryWithResources01 {
2:     public static void main(String[] args) {
3:        try (Scanner sc = new Scanner(System.in)) {
4:           System.out.print("O scanner será encerrado após a execução");
5:        } catch (RuntimeException e) {
6:           System.out.println(e.getMessage());
7:        }
8:     }
9:  }
```

Multi-catch

```
1:  public class TesteMultCatch01 {
2:    public static void main(String[] args) {
3:       File arquivo = new File("c:/output/output.txt");
4:       try (PrintStream ps = new PrintStream(arquivo)) {
5:          System.setOut(ps);
6:          System.out.println("Isso será escrito no arquivo de Output!");
7:       } catch (FileNotFoundException | SecurityException e) {
8:          System.out.println("Verifique suas pemissões, o nome e o caminho do arquivo: " +
9:                e.getMessage());
10:      }
11:   }
12: }
```

Atenção: superclasse e subclass com **multi-catch**

Classes que tem relação de herança não podem ser utilizadas com multi-catch, por exemplo, você não pode fazer um **muti-cath** com **FileNotFoundException** e **IOException**.

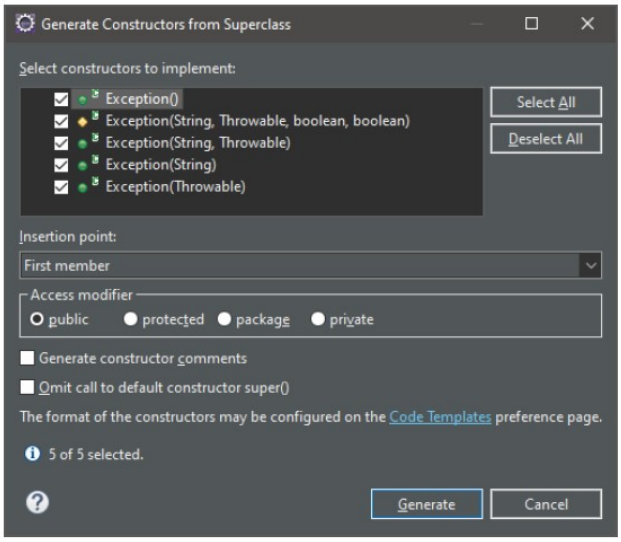
Customizando erros

```
1:  public class ArquivoOutput {
2:    static public void grava(String nomeArquivo, String textoMensagem) throws RuntimeException
    {
3:       File arquivo = new File(nomeArquivo);
4:       try (PrintStream ps = new PrintStream(arquivo)) {
5:          System.setOut(ps);
6:          System.out.println(textoMensagem);
7:       } catch (FileNotFoundException | SecurityException e) {
8:          throw new RuntimeException("Verifique suas pemissões, o nome e o caminho do arquivo:
    " + e.getMessage());
9:       }
10:   }
11: }
```

checked ou unchecked???

```
1:  public class SistemaException extends Exception { //extends RuntimeException => unchecked
2:
3:    public SistemaException() {
4:       super();
5:    }
6:
7:    public SistemaException(String message, Throwable cause, boolean enableSuppression, boolean
    writableStackTrace) {
8:       super(message, cause, enableSuppression, writableStackTrace);
9:    }
10:
11:   public SistemaException(String message, Throwable cause) {
12:      super(message, cause);
13:   }
14:
15:   public SistemaException(String message) {
16:      super(message);
17:   }
18:
19:   public SistemaException(Throwable cause) {
20:      super(cause);
21:   }
22: }
```

checked (try/catch)



A **`String message`** que é recebida nos construtores é a mesma retornada pelo método **`getMessage()`**. Podemos então instanciar ao erro com a informação que queremos que seja exibida para o usuário da classe.

```
1:  public class TesteSistemaException01 {
2:     public static void main(String[] args) throws SistemaException {
3:        throw new SistemaException("Não foi possivel gravar dado no BD.");
4:     }
5:  }
```

> **Atenção**: Sobrescrevendo métodos que lançam erro
>
> Quando um método que lança algum tipo de erro é sobrescrito na subclass, você pode engolir o erro e relançá-lo usando alguma subclass do tipo do erro original, nunca uma superclasse.

Assertions

`assert booleanExpression;`

- Esta declaração testa a expressão booleana.
- Não faz nada se a expressão booleana for avaliada como **verdadeira**.
- Se a expressão booleana for avaliada como **falsa**, esta declaração lança um **AssertionError**.

`assert booleanExpression: expression;`

- Funciona como o `assert booleanExpression ;`.
- Além disso, se a expressão booleana for avaliada como falsa, o segundo argumento é convertido em uma String e é usado como texto descritivo na mensagem **AssertionError**.

```
1:  public class TesteAssert01 {
2:    public static void main(String[] args) {
3:       check(-5);
4:
5:    }
6:
7:    static public void check(int num) {
8:       if (num > 0) {
9:          System.out.println("O número é positivo!");
10:      } else if (num == 0) {
11:         System.out.println("O número é igual a zero!");
12:      } else {
13:         assert (num > 0);
14:      }
15:   }
16: }
```

falso para número negativo

```
1:  public class TesteAssert02 {
2:    public static void main(String[] args) {
3:       check(-5);
4:
5:    }
6:
7:    static public void check(int num) {
8:       if (num > 0) {
9:          System.out.println("O número é positivo!");
10:      } else if (num == 0) {
11:         System.out.println("O número é igual a zero!");
12:      } else {
13:         assert (num > 0) : "O valor é negativo!";
14:      }
15:   }
16: }
```

```
java -enableassertions MeuPrograma
java -ea MeuPrograma
```

9.2 LocalDate, LocalTime e LocalDateTime

`LocalDate.now()`:

- Data atual

`LocalDate.of(ano, mês, dia)`:

- Gera uma data a partir dos argumentos informados

```
1:  public class TesteLocalDate01 {
2:     public static void main(String[] args) {
3:        LocalDate hoje = LocalDate.now();
4:        LocalDate futuro = hoje.plusYears(1).plusMonths(5).plusDays(10);
5:        System.out.println(Formatador.data(futuro));
6:
7:        Period period = Period.of(0, 1, 5);
8:        futuro = hoje.plus(period);
9:        System.out.println(Formatador.data(futuro));
10:    }
11: }
12: /*
13:  Output:
14:  terça-feira, 4 de abril de 2023
15:  terça-feira, 30 de novembro de 2021
16: */
```

A classe **`java.time.LocalTime`** armazena apenas a hora. Podemos obter uma hora usando os seguintes construtores:

`LocalTime.now()`:

- Hora atual

`LocalTime.of(hora, minuto, segundo)`:

```
import static java.time.temporal.ChronoUnit.*;
1:
2:  public class TesteocalTime01 {
3:     public static void main(String[] args) {
4:        LocalTime inicioAula = LocalTime.of(18, 30);
5:        Duration duracao = Duration.ofHours(4).plusMinutes(10);
6:        LocalTime fimDaAula = inicioAula.plus(duracao);
7:        System.out.println(fimDaAula);
8:
9:        LocalTime horaAtual = LocalTime.now();
10:       long minutosParaFim = horaAtual.until(fimDaAula, MINUTES);
11:       System.out.println(minutosParaFim);
12:    }
13: }
```

`LocalDateTime.now()`:

- Data e hora atual

`LocalDateTime.of(LocalDate, LocalTime)`:

- Gera uma data a partir dos argumentos informados

`LocalDateTime.of(ano, mês, dia, hora, minuto, segundo)`:

- Gera uma data a partir dos argumentos informados

```
import static br.com.kasolution.ferramenta.Formatador.data;
import static java.time.Month.*;
1:
2:  public class TesteLocalDateTime01 {
3:    public static void main(String[] args) {
4:       LocalDate anoNovo = LocalDate.of(2022, JANUARY, 1);
5:       LocalTime meiaNoite = LocalTime.of(0, 0);
6:       LocalDateTime reveillon  = LocalDateTime.of(anoNovo, meiaNoite);
7:       LocalDateTime fimFerias = reveillon.plusDays(10).plusHours(7);
8:       System.out.printf("Fim das ferias: %s \n", data(fimFerias));
9:    }
10: }
```

```
ZoneId saoPaulo = ZoneId.of("America/Sao_Paulo");
ZoneId italia = ZoneId.of("Europe/Rome");
ZoneId japao = ZoneId.of("Asia/Tokyo");
```

```
ZoneOffset saoPaulo = ZoneOffset.of("-3");
ZoneOffset nepal = ZoneOffSet.ofHousMinutos(5, 45);]
```

```
import static br.com.kasolution.ferramenta.Formatador.*;
import static java.time.Month.*;
1:
2:  public class TesteZoneId01 {
3:    public static void main(String[] args) {
4:       ZoneId saoPaulo = ZoneOffset.of("-3");
5:       ZoneId vancouver = ZoneId.of("America/Vancouver");
6:       LocalDateTime data = LocalDateTime.of(2021, OCTOBER, 18, 18, 30);
7:       ZonedDateTime aulaBrasil = ZonedDateTime.of(data, saoPaulo);
8:       ZonedDateTime aulaCanada = aulaBrasil.withZoneSameInstant(vancouver);
9:
10:      System.out.println("Aula no Brasil: " + data(aulaBrasil));
11:      System.out.println("Aula no Canadá: " + data(aulaCanada));
12:   }
13: }
```

9.4 Porções de Data e Hora

```
1:  public class TesteInstant01 {
2:    public static void main(String[] args) {
3:       LongStream numeros = LongStream.generate(() -> (int) (Math.random() * 1000));
4:
5:       System.out.printf("Antes: %s \n", Formatador.data(Instant.now()));
6:       numeros.limit(100_000_000)
7:          .sorted()
8:          .sum();
9:       System.out.printf("Depois: %s \n", Formatador.data(Instant.now()));
10:   }
11: }
```