

# Exceções

---

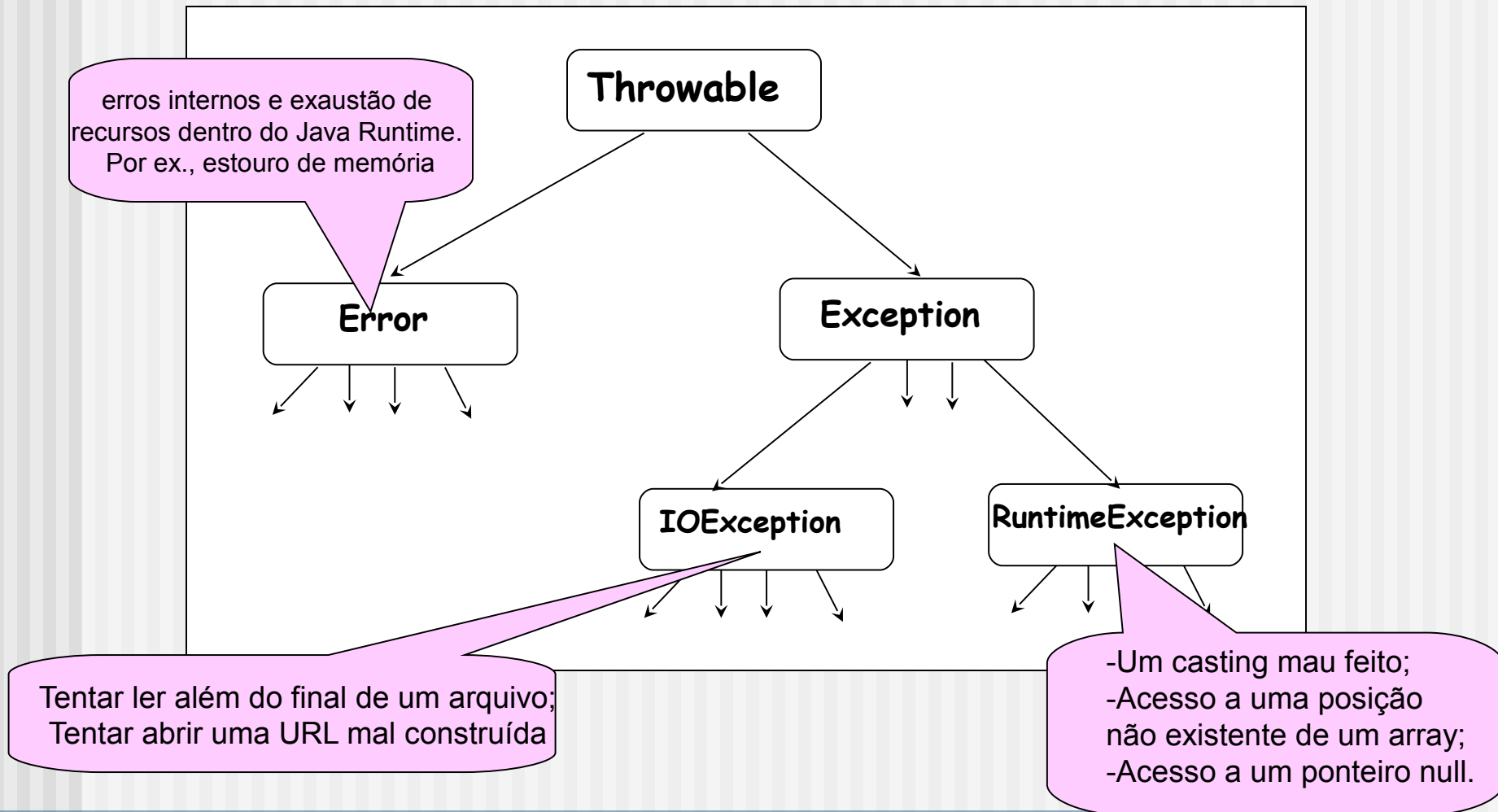
Criando suas  
exceções

# Tratamento de Exceções

---

- O tratamento de exceções em Java permite o gerenciamento de erros em tempo de execução.
- Uma exceção em Java é um objeto que descreve uma condição de exceção que ocorreu em algum fragmento de código.
- Quando surge uma condição excepcional, um objeto *Exception* é criado e lançado no método que causa a exceção.

# Classificação de tipos de Exceções



# Exemplo

```
class ExcecaoDivisaoPorZero {  
    public static void main (String args []) {  
  
        int d=0;  
        int a=42/d;  
        System.out.println ("Execução continua.");  
    }  
}
```

## Output



```
c:\jdk1.3\bin\java.exe ExcecaoDivisaoPorZero  
java.lang.ArithmeticException: / by zero  
    at ExcecaoDivisaoPorZero.main(ExcecaoDivisaoPorZero.java:6)  
Exception in thread "main"
```

O programa pára a execução e é chamado o manipulador de exceção padrão que mostra o tipo de exceção, método em que ela aconteceu, nome o arquivo e linha.

# try e catch

```
class ExcecaoDivisaoPorZero    {  
    public static void main (String args []) {  
        try{  
            int d=0;  
            int a=42/d;  
            System.out.println ("Dentro do bloco da exceção.");  
        }  
        catch (ArithmeticException e) {  
            System.out.println ("Aconteceu divisão por zero.");  
        }  
        System.out.println ("Execução continua.");  
    }  
}
```

## Output



c:\jdk1.3\bin\java.exe ExcecaoDivisaoPorZero  
Aconteceu divisão por zero.  
Execução continua.



# Várias Cláusulas Catch

```
class MultiCatch {  
    public static void main (String args [ ]) {  
        try{  
            int a = args.length;  
            System.out.println ("a = "+a);  
            int b = 42/a;  
            int c [ ] = {1};  
            c[42] = 99;  
        }  
        catch (ArithmeticException e) {  
            System.out.println ("Div por 0: "+e);  
        }  
        catch (ArrayIndexOutOfBoundsException e) {  
            System.out.println ("Estouro indice array: "+e);  
        }  
        System.out.println ("Execução continua.");  
    }  
}
```

# Criando a minha classe de exceções

---

```
class MinhaExcecao extends Exception {  
    private int detalhe;  
  
    public MinhaExcecao(int a) {  
        detalhe = a;  
    }  
  
    public String toString() {  
        return "MinhaExcecao [" + detalhe + "];"  
    }  
} // da class MinhaExcecao
```

# Usando throw para lançar exceções

---

```
class DemoExcecao {  
    public static void main(String args[]) {  
        try {  
            int a = 11;  
            if (a > 10) {  
                MinhaExcecao minhaExc = new MinhaExcecao(a);  
                throw minhaExc;  
            }  
        } catch (MinhaExcecao e) {  
            System.out.println("Excecao capturada: " + e);  
        }  
    }  
} // da class DemoExcecao
```



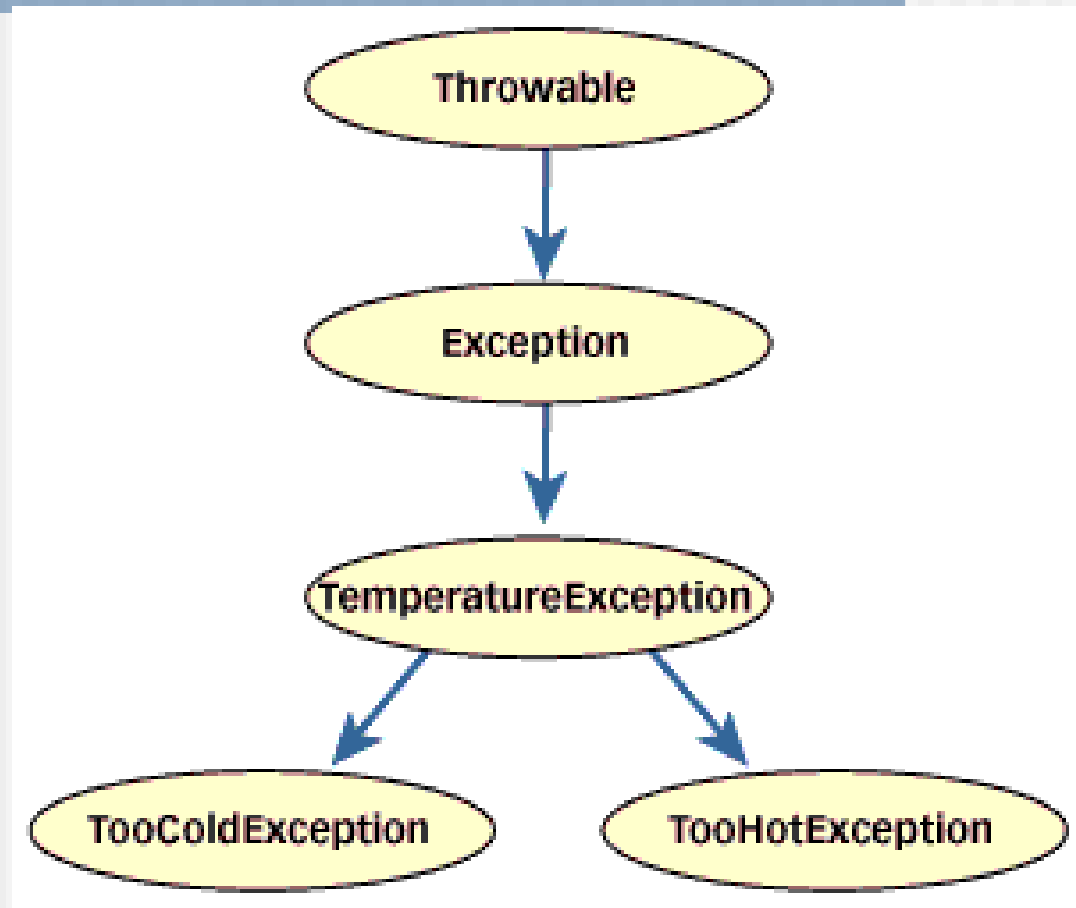
# throws

---

```
class DemoThrows {
    public static void proced () throws MinhaExcecao{

        System.out.println ("No Procedimento.");
        throw new MinhaExcecao (1);
    }
    public static void main (String args []) {
        try {
            proced ();
        }
        catch (MinhaExcecao e) {
            System.out.println("Aconteceu exceção do"+
            "tipo MinhaExcecao.");
        }
    }
}
```

# Criando a minha classe de exceções



# Criando a minha classe de exceções

```
class TemperatureException extends Exception {  
    public String toString() {  
        return "Tem algum problema com a temperatura!";  
    }  
}  
  
class TooColdException extends TemperatureException {  
    public String toString() {  
        return "A temperatura está gelada demais!";  
    }  
}  
  
class TooHotException extends TemperatureException {  
    public String toString() {  
        return "A temperatura está quente demais!";  
    }  
}
```

# Gerando exceções

```
class VirtualPerson {
    private static final int tooCold = 65;
    private static final int tooHot = 85;

    public void drinkCoffee(CoffeeCup cup) throws TooColdException, TooHotException
    {
        int temperature = cup.getTemperature();
        if (temperature <= tooCold) {
            throw new TooColdException();
        } else if (temperature >= tooHot) {
            throw new TooHotException();
        }
        // ...
    }
    // ...
}

class CoffeeCup {
    private int temperature = 75;
    public void setTemperature(int val) {
        temperature = val;
    }
    public int getTemperature() {
        return temperature;
    }
    // ...
}
```

---

```
class Excecao2 {
    public static void main(String[] args) {
        int temperature = 0;
        if (args.length > 0) {
            try {
                temperature = Integer.parseInt(args[0]);
            }
            catch (NumberFormatException e) {
                System.out.println(
                    "Tem que passar um inteiro como argumento.");
                return;
            }
        }
        else {
            System.out.println(
                "Tem que passar uma temperatura.");
            return;
        }
    }
}

// continua ...
```

---

```
// Criando copo de café
CoffeeCup cup = new CoffeeCup();
cup.setTemperature(temperature);
// cria um cliente
VirtualPerson cust = new VirtualPerson();
try {
    // cliente bebe café
    cust.drinkCoffee(cup);
    System.out.println("Coffee is just right.");
}
catch (TooColdException e) {
    System.out.println("Coffee is too cold.");
    //lidar com cliente muito bravo! :-)
}
catch (TooHotException e) {
    System.out.println("Coffee is too hot.");
    //lidar com cliente muito bravo! :-)
}
}
```

```
}
```

# Exercício

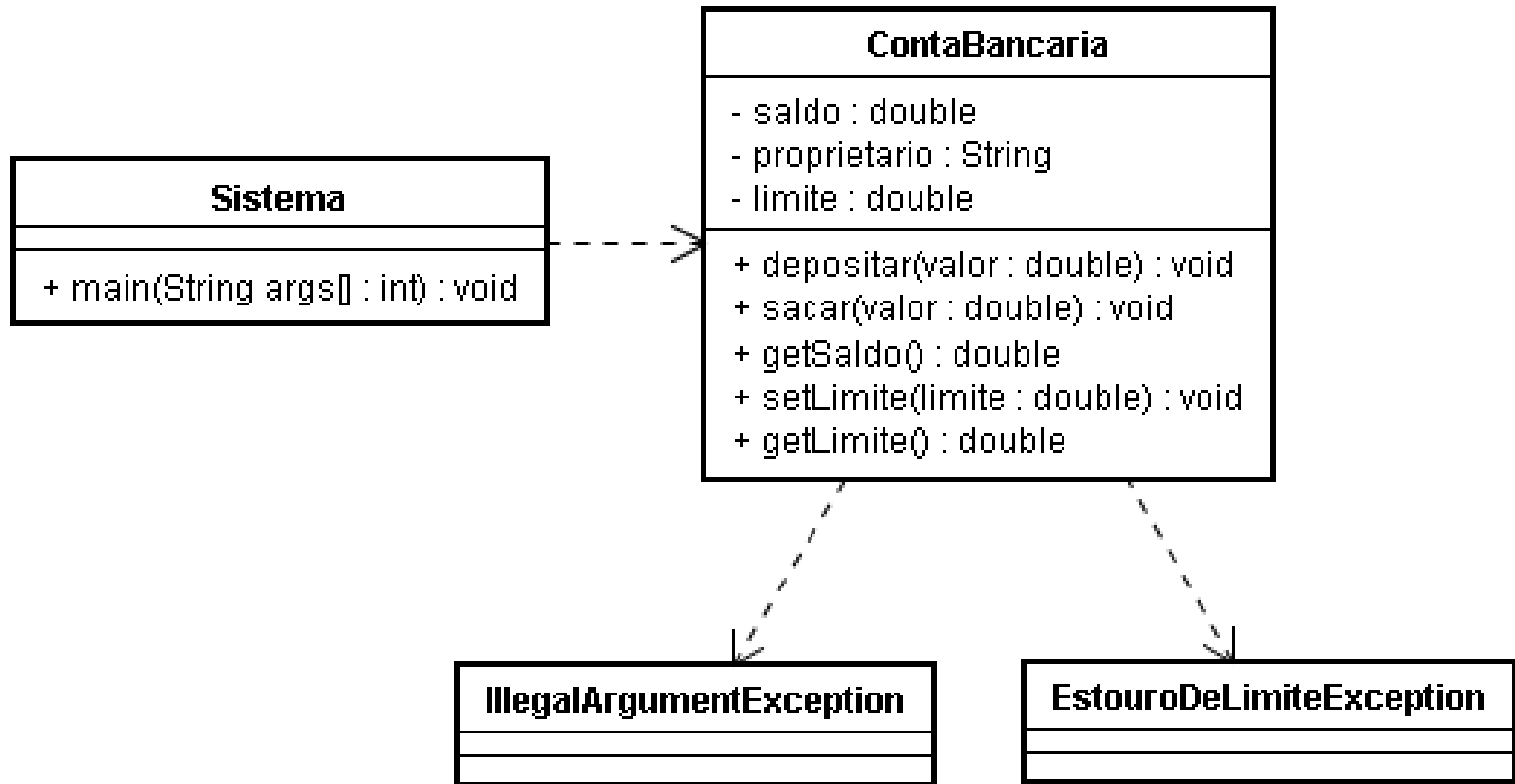
1. Faça um programa para somar dois números:
  - Faça um função **"int obterIntValido()"** para receber número inteiro válido.
  - Enquanto não for válido, a função permanece no loop solicitando e exibindo a mensagem de erro.
  - Faça o programa principal chamá-la para receber **n** e **n1**.
2. Escreva um programa que
  - armazene em um vetor os nomes dos meses do ano
  - solicite ao usuário que digite um valor inteiro.
  - mostre o nome do mês correspondente ao número digitado.
  - OBS: tratar com exceções a digitação inválida e o índice do mês inválido.
3. Altere o programa para que fique rodando iterativamente e que finalize quando for digitado o valor 99 para o mês.

# Exercício 2

4. Implemente uma classe `ContaBancaria`. Esta classe deve ter como atributo interno a informação sobre o saldo do `ContaBancaria` (considere um valor `double`), o nome do proprietário da conta e um limite de crédito.
5. Implemente, além dos necessários métodos construtores, um método `Depositar`, um método `Sacar`, um método para informar o saldo atual e demais métodos necessários. Garanta, utilizando mecanismo de exceção, que não será depositado um valor negativo na conta para depósito, utilizando a exceção `IllegalArgumentException`. Já para o método `sacar`, garanta que não seja retirado um valor além do limite da conta com `EstouroDeLimiteException` e, também, que não seja informado um saque negativo, utilizando a `IllegalArgumentException`.
6. Crie um classe principal (`Sistema`) e no `main` crie um objeto da classe `ContaBancaria` solicite ao usuário o nome do correntista e qual será o seu limite.
7. Depois, enquanto o usuário desejar, solicite qual operação ele deseja realizar (depósito, saque, saldo ou sair). Realize o tratamento de exceções correspondente a cada operação realizada.



# Exercício (continuação)

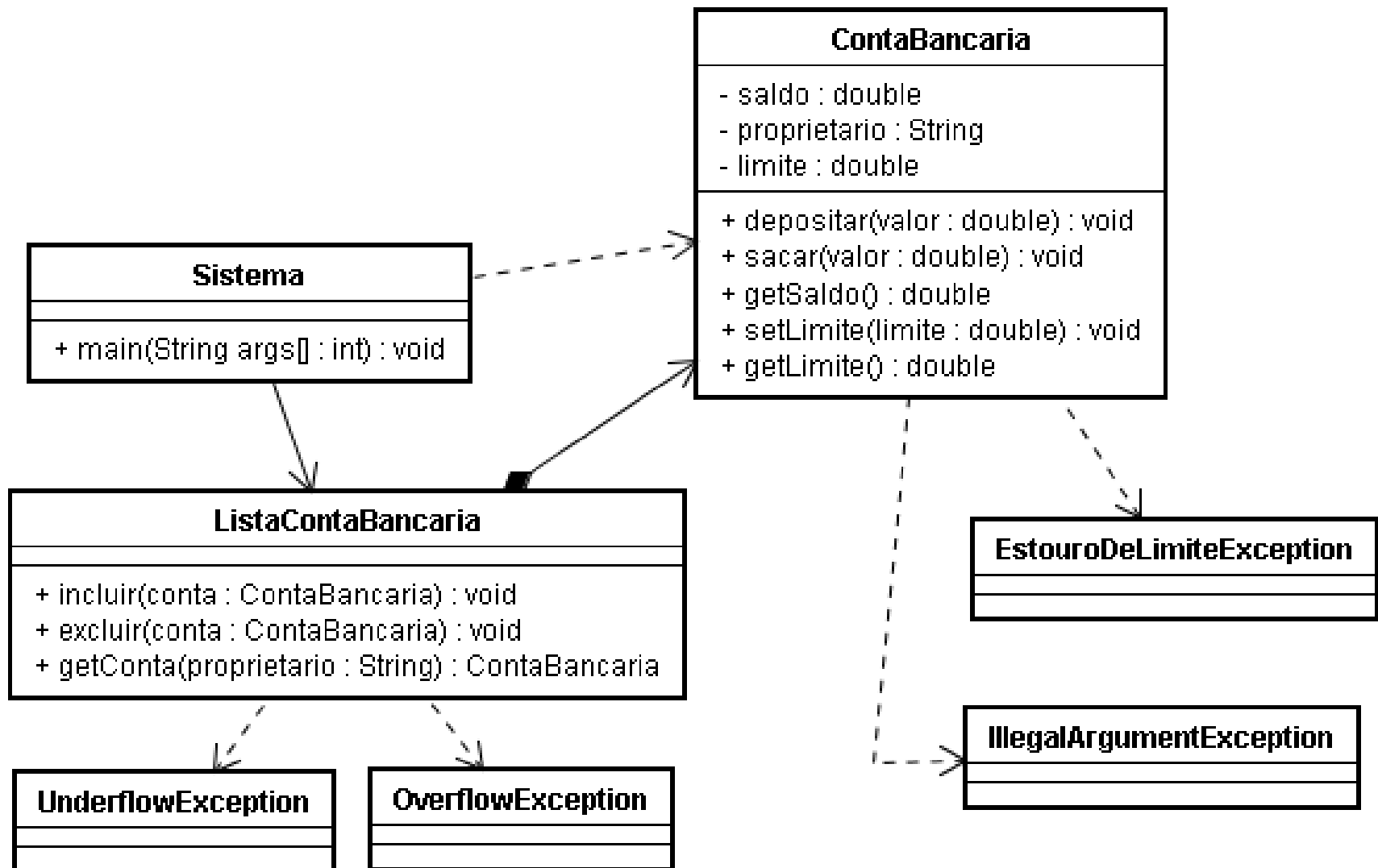


# Exercício 2

---

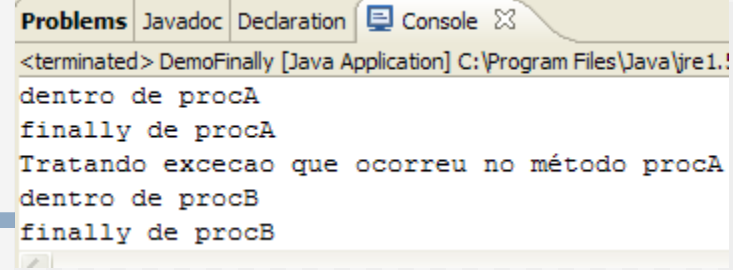
- Continuando o exercício anterior, crie a classe `ListaContaBancaria`, que deve manter uma lista de contas bancárias. Para tanto, a classe fornece métodos para incluir, excluir e obter uma conta bancária da lista.
  - Esta classe pode gerar a exceção `OverflowException`, quando o usuário deseja incluir uma conta e a lista já está no máximo de registros. `UnderflowException`, quando o usuário tentar remover um registro e a lista está vazia. Por fim, pode gerar `NotFoundException`, quando o usuário tentar buscar uma conta que não existe.
  - Altere a classe `Sistema` para permitir o cadastramento de diversas contas (incluir, alterar, excluir e consultar dados da conta).
  - Por fim, antes de realizar cada operação, deve-se informar de qual conta bancária está se querendo depositar, sacar ou ver saldo.
-

# Exercício 2



# finally

```
class DemoFinally {
    static void procA() throws Exception {
        try {
            System.out.println("dentro de procA");
            throw new Exception("demo");
        } finally {
            System.out.println("finally de procA");
        }
    }
    static void procB() {
        try {
            System.out.println("dentro de procB");
            return;
        } finally {
            System.out.println("finally de procB");
        }
    }
    public static void main(String args[]) {
        try {
            procA();
        } catch (Exception e) {
            System.out.println("Tratando excecao que ocorreu no método procA");
        }
        procB();
    } } // da class
```



Problems Javadoc Declaration Console

<terminated> DemoFinally [Java Application] C:\Program Files\Java\jre1.6\bin\java.exe  
dentro de procA  
finally de procA  
Tratando excecao que ocorreu no método procA  
dentro de procB  
finally de procB

# Utilizando printStackTrace

- Método herdado da Classe Throwable
- Imprime lista de chamada de métodos

```
1 class UsandoprintStackTrace {
2     public static void main(String args[]) {
3         try{
4             metodo1();
5         }
6         catch (Exception e) {
7             System.out.println("Erro:" + e);
8             e.printStackTrace();
9         }
10    }
11    public static void metodo1() throws Exception {
12        metodo2();
13    }
14    public static void metodo2() throws Exception {
15        metodo3();
16    }
17    public static void metodo3() throws Exception {
18        throw new Exception();
19    }
20 }
```

```
Erro:java.lang.Exception
java.lang.Exception
    at UsandoprintStackTrace.metodo3(TestaPrintStackTrace.java:18)
    at UsandoprintStackTrace.metodo2(TestaPrintStackTrace.java:15)
    at UsandoprintStackTrace.metodo1(TestaPrintStackTrace.java:12)
    at UsandoprintStackTrace.main(TestaPrintStackTrace.java:3)
```

# Para saber mais:

---

- <http://www.javaworld.com/javaworld/jw-07-19>
  - <http://java.sun.com/docs/books/tutorial/essen>
-

# Exercício Polimorfismo

---

- Implemente a hierarquia de formas geométricas (quadrado, círculo, losango, retângulo e triângulo, esfera, cubo, pirâmide..). Considere que as formas devem ter um nome, coordenadas x e y (para formas bidimensionais), e x, y e z para formas tridimensionais.
- Cada forma bidimensional deve conter o método obterArea(), e cada forma tridimensional deve conter o método obterArea() e obterVolume().
- Crie um programa que utilize um vetor de formas para objetos dessa hierarquia. Apresente um menu de opções para o usuário, de forma que ele digite (1) para criar um círculo, (2) para criar um quadrado, e assim por diante. O programa deve mostrar, a qualquer momento, uma descrição textual do objeto ao qual cada elemento se refere, considerando as formas já inseridas. O programa também deve percorrer o vetor e mostrar o valor da área de todas as formas cadastradas.