Programação II Casting, Try Catch

Ficha(s) Pratica(s)	*10*

Em java não existe suporte a atribuições arbitrárias entre variáveis de tipos diferentes, por exemplo não pode-se inicializar uma variável inteira e atribui-lhe um valor decimal sem indicar de forma explícita o processo a usar. Veja o código abaixo:

```
3 public class Casting {
4     public static void main(String[] args) {
5         int numeroInteiro;
6         numeroInteiro = 2.6;
7     }
8 }
```

Para resolver isso usamos um processo chamado **casting**. Quando atribuímos um valor a uma variável, e o valor passado é incompatível com o tipo de dado definido, ocorrerá uma conversão. Para alguns casos, essa conversão será automático e em outros casos o programador deve indicar de que forma o valor será convertido ao tipo de dado da variável.

```
3 public class Casting {
4     public static void main(String[] args) {
5         int numeroInteiro;
6         numeroInteiro = (int) 2.6;
7     }
8 }
```

Quando o **casting** for feito de forma automática, estamos perante um casting implícito. Veja o exemplo abaixo apesar de 10 ser um número inteiro, o tipo de dado da variável numeroLongo continua sendo do tipo `long`.

```
3 public class Casting {
4     public static void main(String[] args) {
5         long numeroLongo = 10;
6     }
7 }
```

Como visto nos exemplos acima, algumas conversões não podem ser realizadas de forma automática, pois o compilador não pode assumir que tipo de conversão ele deve realizar, isso acontece porque o tamanho de tipo de dado a ser recebido por uma variável é maior que o tamanho pré-definido para o tipo dessa variável, logo o compilador não sabe como ajustar os bits que excederam.

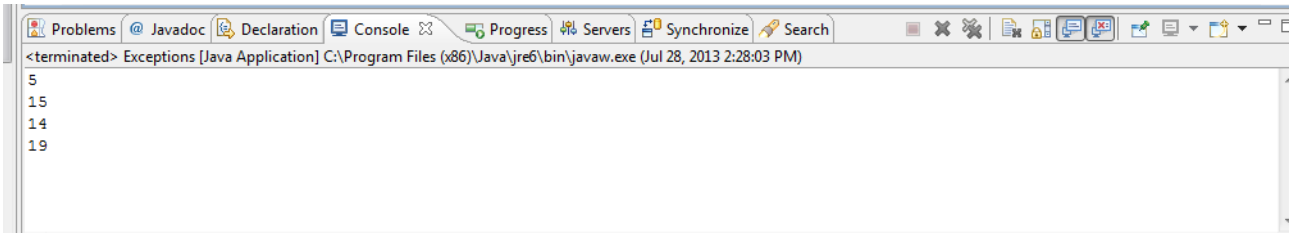
Introdução as Exceções try catch

Para entendermos melhor esse assunto crie um classe e dentro dela inicialize um array do tipo inteiro com 4 posições preenchidas de forma aleatória, percorra o array com ajuda de um ciclo for imprimindo cada valor. Veja o código abaixo:

```
public class Exceptions {
	public static void main(String[] args) {
		int[] numeros = new int[4];
		numeros[0]=5;
		numeros[1]=15;
		numeros[2]=14;
		numeros[3]=19;

		for (int i = 0; i < numeros.length; i++) {
			System.out.println(numeros[i]);
		}
	}
}
```

O resultado da execução desse programa deve ser:

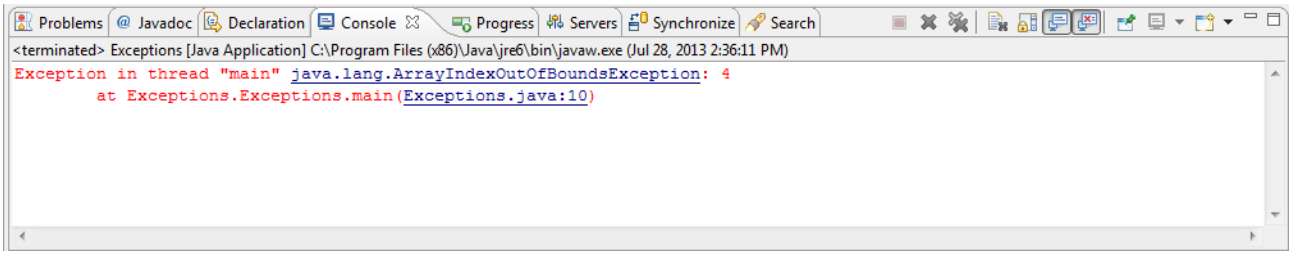


Agora imagine que um programador por engano coloca-se mais um elemento dentro do array, sem notar que o array já esta cheio, ou seja sem perceber que o array já foi preenchido por completo.

```
public class Exceptions {
	public static void main(String[] args) {
		int[] numeros = new int[4];
		numeros[0]=5;
		numeros[1]=15;
		numeros[2]=14;
		numeros[3]=19;
		numeros[4]=19; //esta posição não existe

		for (int i = 0; i < numeros.length; i++) {
			System.out.println(numeros[i]);
		}
	}
}
```

Execute o programa e veja o resultado. Uma exceção será lancada do tipo NULLPOINTEXCEPTION, veja o erro abaixo:



Será que o programa deu erro e continuou a execução das outras instruções? Ou ele deu erro e parou todo programa, faca um teste pessoalmente colocando uma instrução depois do ciclo for, o código ficaria assim:

```
public class Exceptions {
	public static void main(String[] args) {
		int[] numeros = new int[4];
		numeros[0]=5;
		numeros[1]=15;
		numeros[2]=14;
		numeros[3]=19;
		numeros[4]=19;

		for (int i = 0; i < numeros.length; i++) {
			System.out.println(numeros[i]);
		}
		System.out.println("consegui executar esse código");
	}
}
```

Ira notar que depois de executar o programa acontece um erro e depois nunca mais é impressa a frase `consegui executar esse código`. Imagina o embaraço que isso poderia criar em um programa que contem vários módulos onde uma empresa uma todos módulos diariamente, modulo de contabilidade, de gestão, de recursos humanos, e apenas um modulo numa certa manha acontece um erro em um no modulo de contabilidade e o programa da uma exceção e como o programador não tratou a exceção todo programa já não funcionada, o que quer dizer que a área de gestão e recursos humanos não pode trabalhar.

Uma analogia pode ser feita, imagina que uma PT eletricidade de Moçambique queima na zona de **Maxaquene A** na cidade de Maputo e como resultado disso a energia de Moçambique toda ficou comprometida, todos ficamos sem energia. Isso é algo que hoje em dia não acontece, porque medidas próprias foram tomadas para que quando acontece um erro em uma outra zona, outra zona não seja afetada.

Para isso no próximo ponto iremos aprender como fazer com que um modulo de um programa

quando da um erro outros módulos não sejam efetuados.

O tratamento de exceções em Java visa ter um código mais tolerante a falhas. Existem 3 palavras reservadas que permitem o tratamento de exceções: **try**, **catch** e **finally**. Esses comandos são usados em conjunto, auxiliando o programador para garantir o desenvolvimento de um código mais robusto.

- **try**: todo tipo que código que pode vir a gerar algum tipo de exceção deve ser agregado dentro do bloco **try**, o código que o programador coloca dentro desse bloco passara a ser designado por código protegido.
- **cacth**: aqui coloca-se o bloco alternativo as instruções que estão no **try**, ou seja, se uma das instruções do bloco **try** der uma exceção, então os comandos do bloco **cacth** serão executados no seu lugar.
- **finally**: é um conjunto de instruções que será sempre executado no final de um bloco **try-cacth**. Independente das exceções geradas na execução do bloco **try-cacth** os comandos do bloco **finally** serão executados.

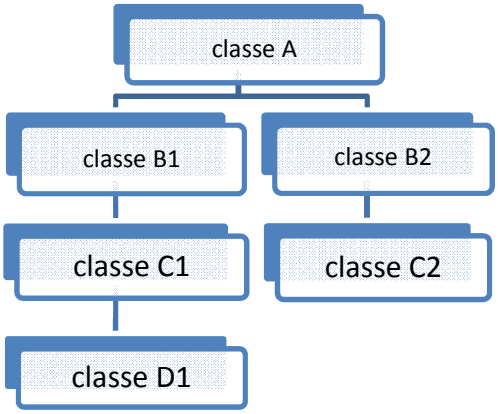
Sintaxe:

```
Public void pesquisaArray(int[] a)
{
	//tratamento de excecoes
	try
	{
		//tenta executar este codigo
		for (int i = 0; i < a.length; i++)
		{
			System.out.println(a[i]);
		}
	}
	catch (ArrayIndexOutOfBoundsException e)
	{
		//se houver algum erro no bloco anterior
		//este comando sera executado
		System.out.println("ERRO " + e.getCause() );
	}
	finally{
		//este comando sera executado sempre
		//Algum codigo que deve ser sempre executado
	}
}
```

Uma dúvida que acaba fazendo os iniciantes em programação não usarem o bloco finally, é saber qual utilidade dar a este bloco. O uso mais comum é quando usamos recursos de entrada e saída do sistema, por exemplo quando estamos a ler um ficheiro externo é por algum motivo o ficheiro gera uma exceção o Java continuaria com uma conexão aberta com ficheiro, neste caso, o bloco finally teria um código de enceramento da conexão com o ficheiro liberando o espaço ocupado, um outro

exemplo similar seria uma conexão a uma base de dados onde não era desejável ter uma conexão aberta depois de ter uma exceção.

Exercício:



Temos a classe "A" que é o topo da hierarquia;

Temos as classes ____ e ____ que estendem a classe "A";

Temos a classe ____ que estende a classe "B1";

Temos a classe ____ que estende a classe "B2";

Temos a classe "D1" que estende a classe ____;

Pela hierarquia conseguimos saber que:

Todo objeto do tipo "B1" ou "B2", é também um objeto do tipo "A";

Todo objeto do tipo "C1" é também um objeto do tipo ____ e por consequência do tipo "A";

Todo objeto do tipo "C2" é também um objeto do tipo ____ e por consequência do tipo ____;

Todo objeto do tipo ____ é também um objeto do tipo "C1" e por consequência do "B1" que é do tipo "A";

O compilador iria executar as linhas de código abaixo:

```
A   a  = new A();
B1 b1 = new B1();
B2 b2 = new B2();
C1 c1 = new C1();
C2 c2 = new C2();
D1 d1 = new D1();
```

--- FIM DO DOCUMENTO ---