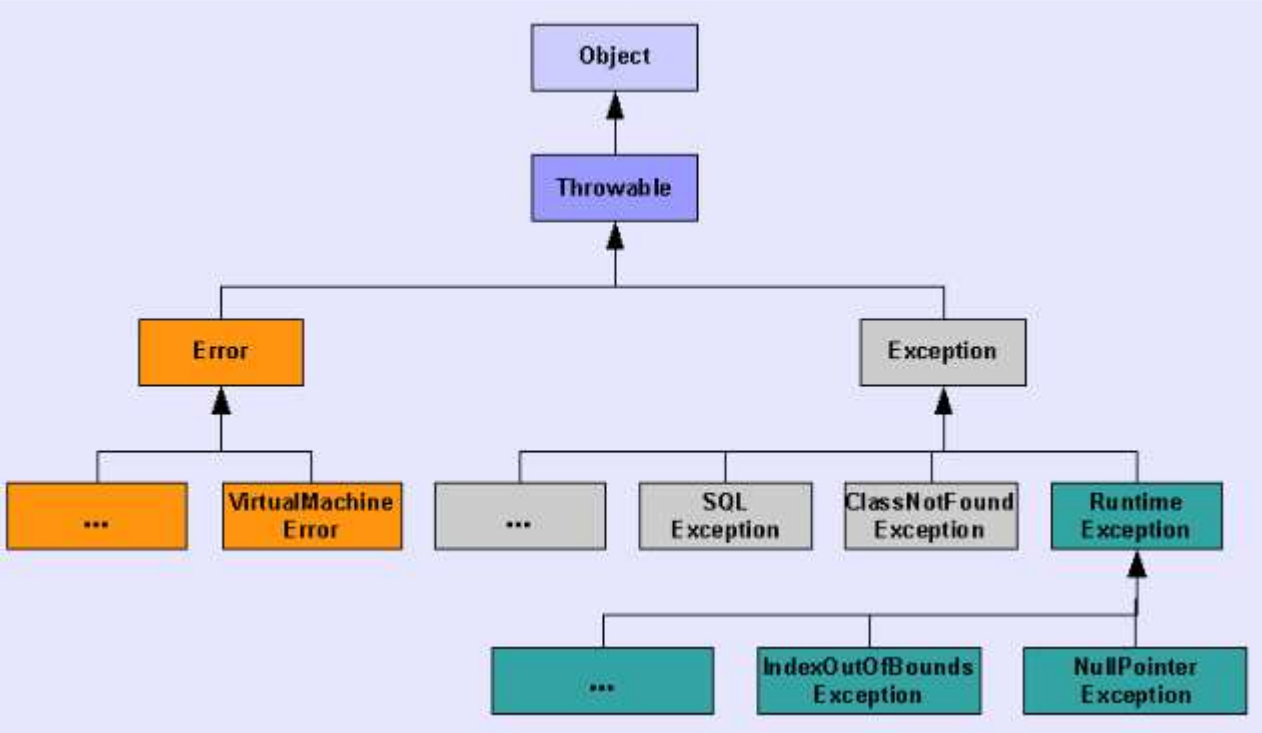
Programação II Exceções

Ficha(s) Pratica(s)	*11*

Depois introduzir-se os comandos para tratamento de exceções: **try**, **catch** e **finally** iremos neste capítulo introduzir a hierarquia de exceções de modo que cada programador conheça quais tipo de exceções possíveis podem ser lançadas, e também como criar suas próprias exceções no caso em que deseja-se lançar uma exceção especifica que não tem na hierarquia apresentada abaixo.



Como pode-se verificar na figura acima todas exceções em Java derivam da classe Throwable e essa classe estende da classe **Object**, assim como qualquer outra classe que não indicamos de forma explícita a sua superclasse.

Ainda na figura acima existem 3 tipos de exceções diferentes: Erros, Exceções previstas e Exceções não previstas. Cada uma delas tem uma função específica e deve ser usada de acordo com a natureza do problema a ser resolvido.

Error: (Erros) Indica uma falha de difícil recuperação durante a execução de um programa, por exemplo tentarmos instanciar uma classe na JVM que já não possui mais memoria livre.

Exception: (Exceções previstas**)** são erros previsíveis, erros causados pelo contexto em que o programa esta a ser executado, por exemplo uma classe não encontrada ou um arquivo não localizado. Estes erros são causados normalmente por falha do utilizador.

Runtime Exception: (Exceções não previstas) é usado para indicar situações em que jamais deveriam acontecer durante a execução de um programa, por exemplo acessar o índice inválido de um array.

Para as **Exceções previstas** o compilador obriga ao programador tratar a exceção antes mesmo da compilação do programa.

Exceções mais comuns

ArrayIndexOutOfBoundsException: tentativa de aceder a uma Posicao inexistente no array.
ClassCastException: tentativa de fazer casting de uma referência que não é classe ou subclasse do tipo desejado.
IllegalArgumentException: argumento formatado de forma diferente do esperado pelo método.
IllegalStateException: o ambiente não permite a execução da operação desejada.
NullPointerException: acesso a um objecto que esta a ser referenciado por uma variável cujo o valor é null;
NumberFormatException: tentativa de converter uma String invalida em um número.
StackOverflowError: quando chamadas recursivas são profundas.
NoClassDefFoundError: a JVM não conseguiu localizar a classe.
ArithmeticException: problemas com operações numéricas, tipicamente divisão por zero.
NegativeArraySizeException: tentativa de criar um array de tamanho negativo.

Lançando Exceções com Throw

Faz-se o uso do **throw** quando deseja-se que em uma determinada situação seja lançada uma exceção para informar que a situação esperada não aconteceu.

Exemplo: Um Aluno possui um método **pagarProprina()**, este método recebe o valor do tipo Double que representa o valor que esta a ser passado para pagar a propina, logo não podemos ter uma situação em que esse valor passado seja um valor negativo, então caso esta situação aconteça, iremos lançar uma exceção do tipo **IllegalArgumentException.**

```
public class Exceptions {

	private double valorAPagar;

	public void pagarPropina(double valor)
	{
	   if(valor < 0)
	   {
			throw new IllegalArgumentException("O valor deve ser maior que
	      0, voce passou o valor: "+ valor);
	   }
	   else
	   {
			valorAPagar = valorAPagar + valor;
	   }
	}
}
```

O resultado dessa operação se o utilizador introduzir um valor menor que zero, será:

```
Exception in thread "main" java.lang.IllegalArgumentException: O valor deve ser
maior que 0, voce passou o valor: -1.0
	at exception.Exceptions.pagarPropina(Exceptions.java:11)
	at exception.TestExceptions.main(TestExceptions.java:7)
```

Criando Exceções propiás

Suponha agora que ao invés de querer lançar uma exceção de `IllegalArgumentException`, deseja-se lançar uma exceção propiá chamada ValorProprinaException, neste caso basta olhar para a hierarquia de exceções e fazer uma extensão de RuntimeException.

```
public class ValorPropinaException extends RuntimeException{
	public ValorPropinaException(String message)
	{
		super(message); //invocando o construtor da superclasse Exception
				//enviando um parametro message
	}
}
```

O método pagar propina deve ficar assim:

```
public class Exceptions {

	private double valorAPagar;

	public void pagarPropina(double valor)
	{
		if(valor < 0)
		{
				throw new ValorPropinaException("O valor deve ser maior
				que 0, você passou o valor: "+ valor);
		}
		else{
			valorAPagar = valorAPagar + valor;
		}
	}
}
```

A execução deste método com o valor negativo, deve resultar na seguinte mensagem:

```
Exception in thread "main" exception.ValorPropinaException: O valor deve ser
maior que 0, voce passou o valor: -1.0
      at exception.Exceptions.pagarPropina(Exceptions.java:11)
      at exception.TestExceptions.main(TestExceptions.java:7)
```

--- FIM DO DOCUMENTO ---