Universidade Presbiteriana Mackenzie
Faculdade de Computação e Informática

Exceções

NOTAS DE AULA

Linguagem de Programação II
2º semestre de 2015
Prof. Tomaz Mikio Sasaki
Versão 1.0

Exceção

- Uma exceção (ou evento excepcional) é um problema que surge durante a execução de um programa.
- Quando uma exceção ocorre, o fluxo normal do programa é interrompido e o programa/aplicação termina de forma anormal.
- As exceções devem ser tratadas para que isso não aconteça.

Referência

As referências para esta aula são:

HORSTMANN, C.S.; CORNELL, G. **Core Java, Volume I - Fundamentals.** Prentice Hall, 2012.

Caelum - Excessões e controles de erros. Disponível em: http://www.caelum.com.br/apostila-java-orientacao objetos/excecoes-e-controle-de-erros

Tutorials Point - Java - Exceptions. Disponível em: http://www.tutorialspoint.com/java/java_exceptions.htm

Observação: As notas de aula são material de apoio para estudo e não têm o objetivo de apresentar o assunto de maneira exaustiva. Não deixe de ler o material de referência da disciplina.

Exemplos de ocorrência de exceções

Uma exceção pode ocorrer por diversos motivos:

- Um usuário inseriu dados inválidos.
- Um arquivo que precisa ser aberto não pode ser encontrado.
- Uma conexão de rede foi perdida no meio da comunicação.
- A JVM está sem memória.

Categorias de exceções

Algumas destas exceções são causados por erro do usuário, outras por erro do programador, e outras por recursos físicos que falharam de alguma maneira.

Com base nisso, temos três categorias de exceções:

- *Checked Exceptions*
- *Unchecked Exceptions*
- *Errors*

Checked Exception

- É aquela que ocorre no tempo de compilação.
- Chamada também de **exceção de tempo de compilação**.
- Não pode ser simplesmente ignorada no momento da compilação. O programador deve ter o cuidado de tratar essa exceção.

Exemplo de *Checked Exception*

```
import java.io.File;
import java.io.FileReader;

public class FilenotFoundDemo {
    public static void main(String args[]){
        File file=new File("E://file.txt");
        FileReader fr = new FileReader(file);
    }
}
```

O programa acima não será compilado com sucesso.

Exemplo de *Checked Exception* (cont.)

```
import java.io.File;
import java.io.FileReader;

public class FilenotFoundDemo {
    public static void main(String args[]){
        File file=new File("E://file.txt");

        // o compilador avisará que esta linha dispara FileNotFoundException
        FileReader fr = new FileReader(file);
    }
}
```

Acesse o javadoc da classe FileReader e note que a assinatura do construtor declara que essa exceção pode ser disparada.

Unchecked Exception

- Também chamada de **exceção de tempo de execução** ou **exceção não verificada**.
- Avisa erros de programação, tais como erros de lógica ou uso indevido de uma API.
- É disparada durante a execução da aplicação.
- Exceções de tempo de execução são ignoradas no momento da compilação.

Exemplo de *unchecked exception*

Se você declara um array de tamanho 5 no seu programa e tenta chamar o sexto elemento do array, ocorre uma exceção do tipo:

- ArrayIndexOutOfBoundsException

Exemplo de unchecked exception (cont.)

```
public class UncheckedDemo {
    public static void main(String args[]) {
        int num[]={1,2,3,4,5};
        System.out.println(num[5]);
    }
}
```

O programa acima será compilado com sucesso.

Exemplo de *unchecked exception* (cont.)

```
public class UncheckedDemo {
    public static void main(String args[]) {
        int num[]={1,2,3,4,5};

        // Na execução desta linha será disparada a exceção
        // ArrayIndexOutOfBoundsException
        System.out.println(num[5]);
    }
}
```

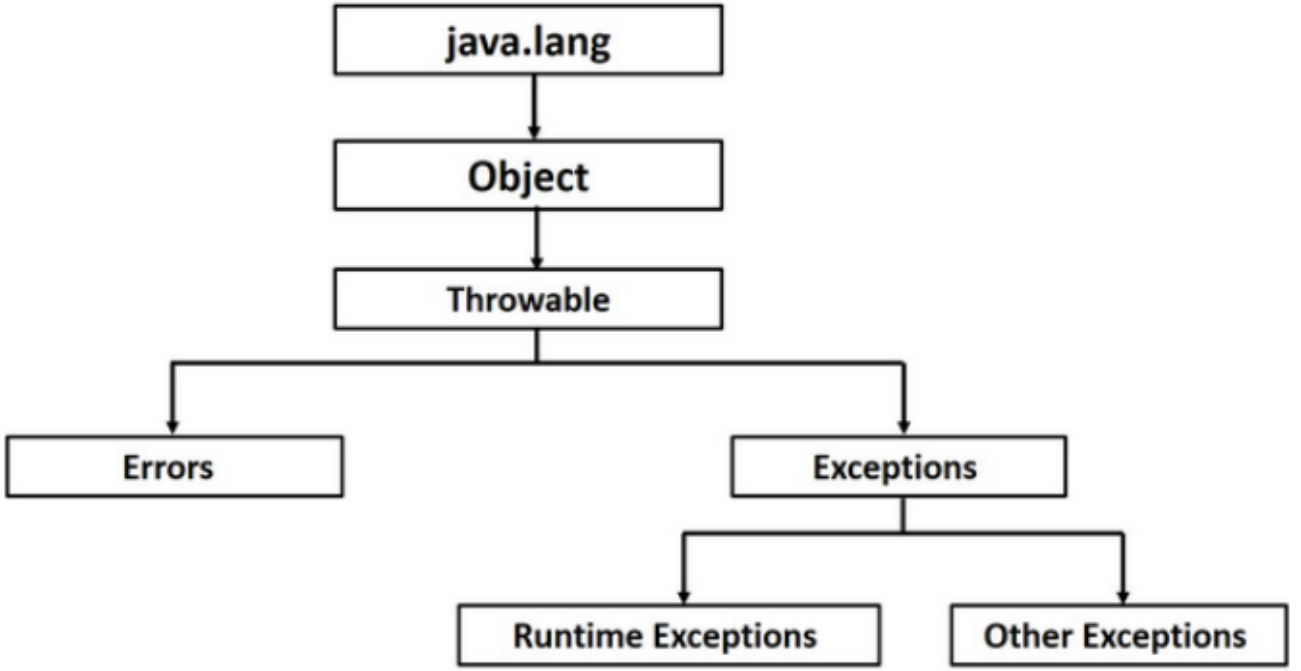
Errors

- Problemas que estão fora do controle do usuário ou programador.
- Não são indicados no momento da compilação.

Exemplo de *error*

- OutOfMemoryError

Hierarquia das exceções



Tratamento de exceções

Se uma exceção ocorre e não é tratada, o programa termina abruptamente e apresenta uma mensagem no console.

Para tratar uma exceção utiliza-se um bloco *try/catch*:

```
try {
    // instruções do fluxo normal
} catch(NomeDaExcecao variavel) {
    // instruções para tratar a exceção
}
```

Exemplo de tratamento de exceção

```java
import java.io.*;

public class ExemploTrataExcecao {
    public static void main(String args[]){
        try{
            int a[] = new int[2];
            System.out.println("Acessando o terceiro elemento :" + a[3]);
        } catch(ArrayIndexOutOfBoundsException e){
            System.out.println("Exceção :" + e);
        }
    }
}
```

Exemplo de tratamento de exceção (cont.)

```
import java.io.*;

public class ExemploTrataExcecao {
    public static void main(String args[]){
        try{
            // instruções do fluxo normal
            int a[] = new int[2];
            System.out.println("Acessando o terceiro elemento :" + a[3]);
        } catch(ArrayIndexOutOfBoundsException e){
            System.out.println("Exceção :" + e);
        }
    }
}
```

Exemplo de tratamento de exceção (cont.)

```
import java.io.*;

public class ExemploTrataExcecao {
    public static void main(String args[]){
        try{
            int a[] = new int[2];
            System.out.println("Acessando o terceiro elemento :" + a[3]);
        } catch(ArrayIndexOutOfBoundsException e){
            // instruções para tratar a exceção
            System.out.println("Exceção :" + e);
        }
    }
}
```

Exemplo de tratamento de mais de uma exceção

```
try {
    file = new FileInputStream(fileName);
    x = (byte) file.read();
} catch(IOException i) { //Erro de leitura
    i.printStackTrace();
    return -1;
} catch(FileNotFoundException f) { //Arquivo não encontrado
    f.printStackTrace();
    return -1;
}
```

Exemplo de tratamento de mais de uma exceção (2)

```
try {
    file = new FileInputStream(fileName);
    x = (byte) file.read();
} catch(IOException | FileNotFoundException ex) {
    //Multicatch
    ex.printStackTrace();
    return -1;
}
```

try/catch/finally

O bloco *finally* segue um bloco *try* ou um bloco *catch*.

Um bloco *finally* de código é sempre executado, independentemente da ocorrência de uma exceção.

```
try {
    // instruções do fluxo normal
} catch(NomeDaExcecao variavel) {
    // instruções para tratar a exceção
} finally {
    // instruções executadas em qualquer situação
}
```

Exemplo de tratamento de exceção com bloco *finally*

```
import java.io.*;

public class ExemploTrataExcecao2 {
    public static void main(String args[]){
        try{
            int a[] = new int[2];
            System.out.println("Acessando o terceiro elemento :" + a[3]);
        } catch(ArrayIndexOutOfBoundsException e){
            System.out.println("Exceção :" + e);
        } finally {
            a[0] = 6;
            System.out.println("First element value: " +a[0]);
            System.out.println("The finally statement is executed");
        }
    }
}
```

try-with-resources (Java 8)

É uma sentença *try* que declara um ou mais **recurso** (*resource*).

O **recurso** é um objeto que deve ser fechado ao sair do bloco *try*.

Exemplo:

```
try (BufferedReader br = new BufferedReader(new FileReader(nomeArquivo))) {
        return br.readLine();
}
```

Bom estudo!