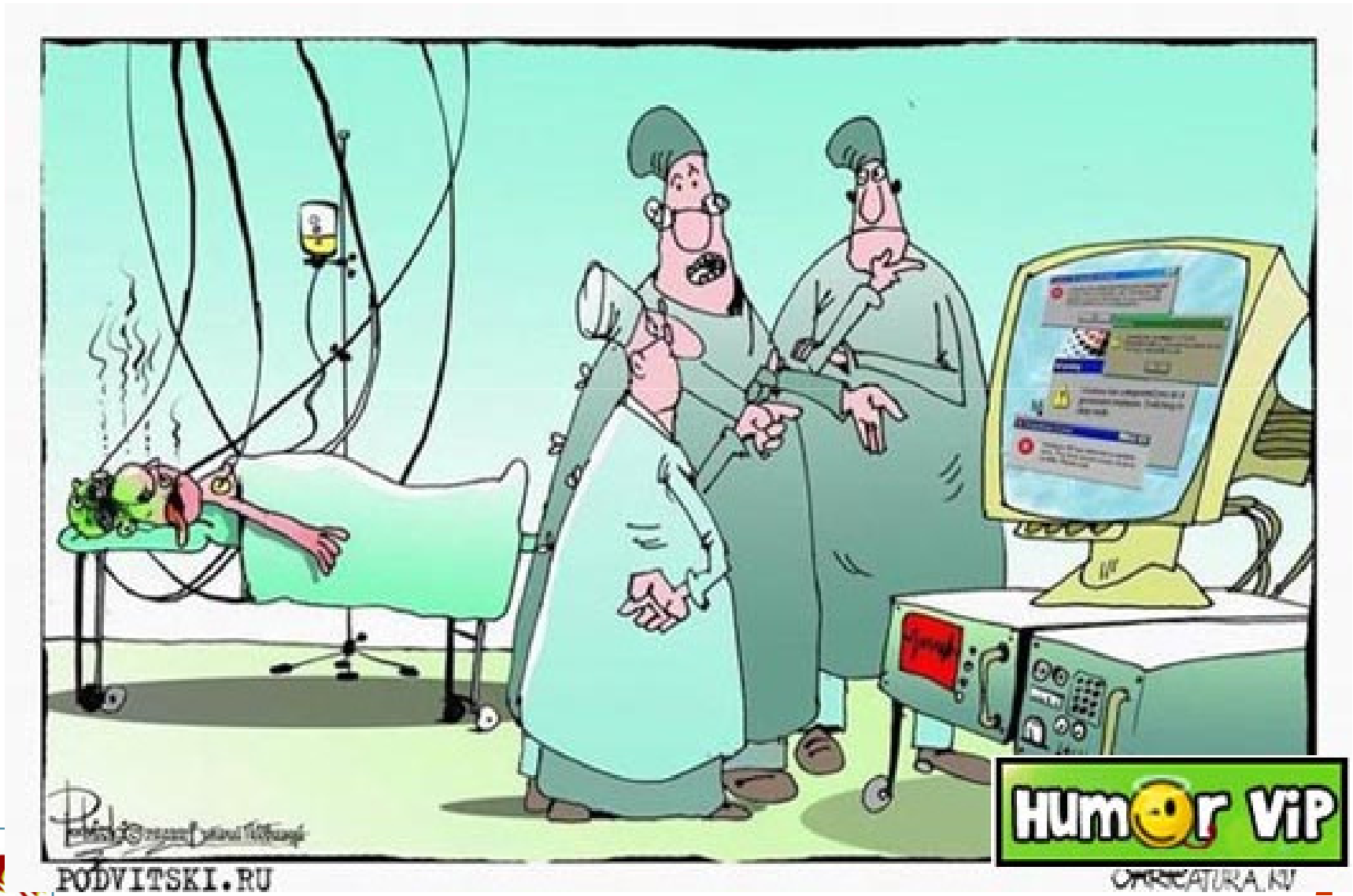
Introdução à Programação

Tratando Exceções

Exceções



Tópicos da Aula

- Hoje vamos aprender como tratar problemas ocorridos durante a execução de um programa
  - Conceito de Exceção
  - Formas de tratar exceções
  - Tratando exceções com *try – catch*
  - Propagação de exceções
- Depois vamos utilizar o conceito de exceção
  - Exemplos em MiniJava com tratamento de exceções
  - Escolhendo onde tratar exceções

Exceções

- Uma **exceção** em Java é um objeto que representa uma situação anormal
- Exceções são **lançadas** por um programa ou ambiente de execução, e podem ser **pegas** e **tratadas** por uma outra parte do programa
- Podemos dizer que um programa é composto de um **fluxo de execução normal** e **um fluxo de execução de uma exceção**
- Um **erro** (error) em Java também é representado como um objeto, porém, geralmente, representa um problema irremediável
  - Portanto, não deve ser tratado

Exemplos de Exceção

- Dividir um número por zero
- Transformar uma String composta de caracteres que não são dígitos em um número
- Abrir um arquivo que não existe
- Invocar métodos de um objeto através de uma referência nula

Exceções em Java

- São objetos comuns, portanto têm que ter uma classe associada
- Classes representando exceções herdam e são subclasses de `Exception` (pré-definida)
- Definem-se subclasses de `Exception` para
  - oferecer informações extras sobre a falha, ou
  - distinguir os vários tipos de falhas

Tratamento de Exceções

- Tratamento de exceção consiste no ato de realizar uma seqüência de comandos quando a exceção ocorre
  - Para tentar contornar o problema
  - Para informar ao usuário a natureza do problema e talvez instruí-lo como o problema pode ser contornado
  - Pode aumentar a **robustez** do programa
- Um programa pode utilizar 3 estratégias para o tratamento de exceções:
  - Simplesmente não tratar a exceção
  - Pegar e tratar a exceção no local onde ela ocorre
  - Propagar e tratar a exceção em outra parte do programa

Tratando Exceções

```
void duplicarValor ( ) {
   int entrada = 0;
   // Antes
  try {
        entrada = campoEntrada.getIntExc();
        resultado = 2 * entrada;
        campoResultado.setInt(resultado);
  }catch (InvalidConversionException e) {
        MiniJavaSystem s = new MiniJavaSystem();
        s.println("Por favor, digite um número");
  }
  //Depois
   campoEntrada.setText("");
}
```

Este método pode lançar uma exceção

Exceção é pega e tratada no corpo do método `duplicarValor()`

Tratando Exceções

- A execução do `try` termina assim que uma exceção é levantada
- O *primeiro* `catch` que tiver uma exceção **compatível** é executado e depois o fluxo de controle passa para o código seguinte ao último `catch`
- Se não houver nenhum `catch` compatível, a exceção e o fluxo de controle são passados para a chamada do código com o `try/catch`

Tratando Exceções: Forma Geral

```
try {...
} catch (Excecao1 e1) {
   ...
}
   ...
} catch (ExcecaoN en) {
   ...
} finally {...}
```

Tratando Exceções

- O bloco `finally` é sempre executado, seja após a terminação normal do `try`, após a execução de um `catch`, ou até mesmo quando não existe nenhum `catch` compatível
- Quando o `try` termina normalmente ou um `catch` é executado, o fluxo de controle é passado para o código seguindo o bloco `finally` (depois deste ser executado)

Como saber quais exceções podem ser levantadas?

- Biblioteca padrão de Java possui uma série de exceções pré-definidas
  - Olhar documentação
- Assinatura de alguns métodos informam as exceções que podem ser levantadas
  - Palavra reservada ***throws***

```
public int getIntExc() throws
InvalidConversionException
```

- Podemos identificar facilmente algumas exceções dependendo dos comandos que realizamos
  - Exemplo: operação de divisão

Propagando Exceções

- Uma exceção pode ser tratada fora do método onde ela ocorreu
- Exceções podem se **propagar** pela hierarquia de métodos até que elas sejam tratadas **ou** quando elas atingem o método *main*
- Todos os tipos de exceções que são lançadas e não tratadas no corpo de um método devem ser declaradas na sua assinatura

Propagando Exceções

duplicarValor()
propaga a exceção

```
void duplicarValor() throws InvalidConversionException
{
  int entrada = 0;
  entrada = campoEntrada.getIntExc();
  resultado = 2 * entrada;
  campoResultado.setInt(resultado);
  campoEntrada.setText("");
}
```

Este método pode lançar uma exceção que não é tratada em duplicarValor()

Exceções *Checked*

- Uma exceção pode ser ***checked*** (checada) ou ***unchecked*** (não checada)
- Diz-se que uma exceção é checked se ela é tratada ou declarada no cabeçalho de um método
- Um erro de compilação ocorrerá, caso uma exceção checked não seja nem tratada nem declarada no cabeçalho de um método

Exceções Un*checked*

- Uma exceção unchecked não requer tratamento, embora ela possa ser tratada
- Em Java, as únicas exceções que são unchecked são objetos do tipo ***`RuntimeException`*** e seus descendentes
  - `ArithmeticException`
  - `NullPointerException`
- Objetos do tipo ***`Error`*** são similares a `RuntimeException`
  - Não devem ser tratados
  - Exemplos: `OutOfMemoryError`, `InstantiationError`, `ThreadDeath`, *etc*

Resumindo ...

- Tratamento de Exceções
  - Exceção X Error
  - Formas de tratar exceções
  - Tratando exceções com *try – catch -finally*
  - Propagação de exceções
  - Exceções Checked e Unchecked
- Exemplos de tratamento de exceções utilizando MiniJava