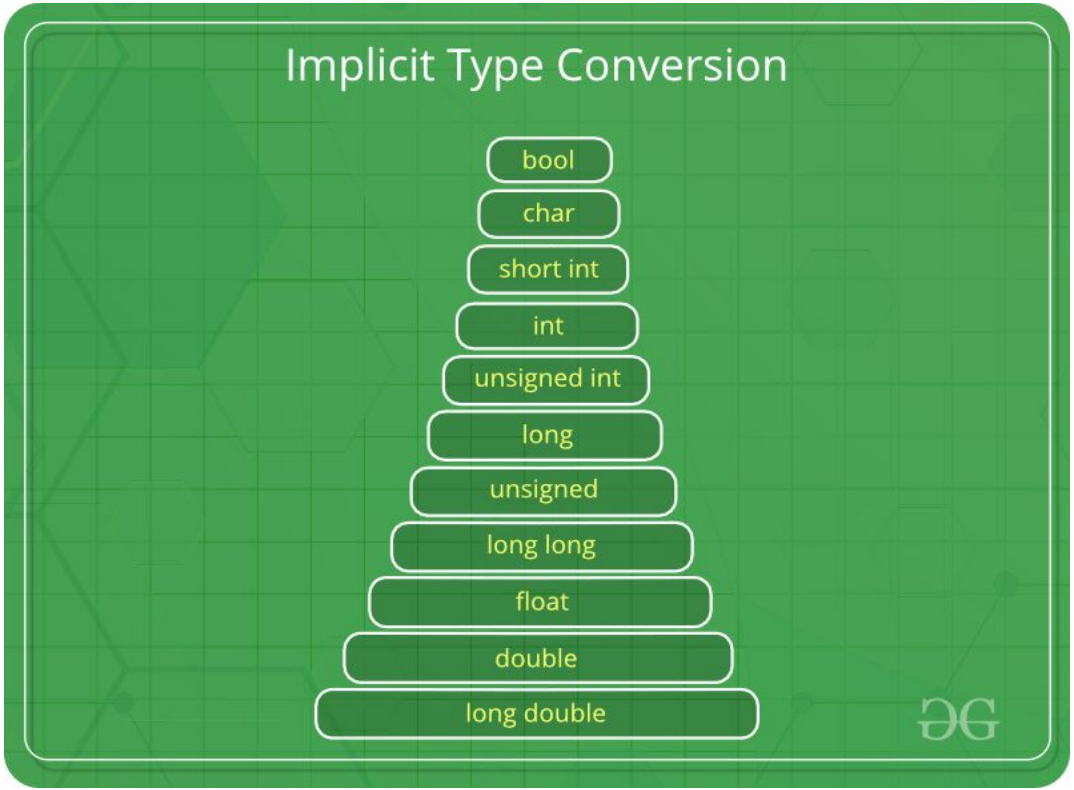
Coerção de Tipos

Lucas A. Lisboa

Conversão de Tipos

- Mudar Tipo de uma Variável
- Explícita ou Implícita (Coerção)
- Menos Abrangente -> Mais Abrangente

Hierarquia de Tipos



Conversão Explícita

- Ocorre quando o programador explicitamente converte um tipo de dado para outro tipo de dado
- Ajuda a evitar erros inesperados, pois o programador está ciente do processo de coerção

Exemplo de Conversão Explícita

- `int("10") = 10`
- `float(5) = 5.0`
- `str(10) = "10"`

Conversão Implícita (Coerção)

- Acontece automaticamente, sem a necessidade de intervenção do programador
- Pode levar a erros inesperados se o programador não estiver ciente do processo de coerção

Exemplo de Coerção

- `5 + 2.5 = 7.5`
- `"10" + 5 = 15`
- `True + 1 = 2`

BNF

ValorConcreto ::= ValorInteiro | ValorBooleano | ValorString | **ValorFloat** | **ValorChar**

ExpUnaria ::= "-" Expressao | "not" Expressao | "length" Expressao

ExpBinaria ::= Expressao "+" Expressao

| Expressao "-" Expressao

| Expressao "and" Expressao

| Expressao "or" Expressao

| Expressao "==" Expressao

| Expressao "++" Expressao

| **Expressao "*" Expressao**

| **Expressao "/" Expressao**

Tipo ::= "string" | "int" | "boolean" | **"float"** | **"char"**

TipoPrimitivos

```
public enum TipoPrimitivo implements Tipo {

    INTEIRO(nome:"INTEIRO"),
    BOOLEANO(nome:"BOOLEANO"),
    STRING(nome:"STRING"),
    FLOAT(nome:"FLOAT"),
    CHAR(nome:"CHAR");
```

```
public boolean eFloat(){
    return this.eIgual(FLOAT);
}

public boolean eChar(){
    return this.eIgual(CHAR);
}
```

ValorFloat

```java
public class ValorFloat extends ValorConcreto<Float> {

    /**
     * Cria <code>ValorFloat</code> contendo o valor fornecido.
     */
    public ValorFloat(Float valor) {
        super(valor);
    }

    /**
     * Retorna os tipos possiveis desta expressao.
     *
     * @param amb
     *            o ambiente de compila��o.
     * @return os tipos possiveis desta expressao.
     */
    public Tipo getTipo(AmbienteCompilacao amb) {
        return TipoPrimitivo.FLOAT;
    }

    public ValorFloat clone(){
        return new ValorFloat(this.valor());
    }
}
```

ValorChar

```java
public class ValorChar extends ValorConcreto<Character> {

    /**
     * Cria <code>ValorChar</code> contendo o valor fornecido.
     */
    public ValorChar(Character valor) {
        super(valor);
    }

    /**
     * Retorna os tipos possiveis desta expressao.
     *
     * @param amb
     *            o ambiente de compila��o.
     * @return os tipos possiveis desta expressao.
     */
    public Tipo getTipo(AmbienteCompilacao amb) {
        return TipoPrimitivo.CHAR;
    }

    public ValorChar clone(){
        return new ValorChar(this.valor());
    }
}
```

ExpMult

```java
public class ExpMult extends ExpBinaria {

    /**
     * Controi uma Expressao de Soma com as sub-expressoes especificadas.
     * Assume-se que estas sub-expressoes resultam em <code>ValorInteiro</code>
     * quando avaliadas.
     * @param esq Expressao da esquerda
     * @param dir Expressao da direita
     */
    public ExpMult(Expressao esq, Expressao dir) {
        super(esq, dir, operador:"*");
    }

    /**
     * Retorna o valor da Expressao de Soma
     */
    public Valor avaliar(AmbienteExecucao amb) throws VariavelNaoDeclaradaException, VariavelJaDeclaradaException {
        return new ValorInteiro(
            ((ValorInteiro) getEsq().avaliar(amb)).valor() *
            ((ValorInteiro) getDir().avaliar(amb)).valor() );
    }
```

ExpDiv

```java
public class ExpDiv  extends ExpBinaria {

    /**
     * Controi uma Expressao de Soma com as sub-expressoes especificadas.
     * Assume-se que estas sub-expressoes resultam em <code>ValorInteiro</code>
     * quando avaliadas.
     * @param esq Expressao da esquerda
     * @param dir Expressao da direita
     */
    public ExpDiv(Expressao esq, Expressao dir) {
        super(esq, dir, operador:"/");
    }

    /**
     * Retorna o valor da Expressao de Soma
     */
    public Valor avaliar(AmbienteExecucao amb) throws VariavelNaoDeclaradaException, VariavelJaDeclaradaException {
        return new ValorInteiro(
            ((ValorInteiro) getEsq().avaliar(amb)).valor() /
            ((ValorInteiro) getDir().avaliar(amb)).valor() );
    }
```

Parser - Tokens

```
TOKEN : /* TOKENS DOS POSS?VEIS TIPOS */
{
  < INT : "int" >
| < BOOLEAN  : "boolean" >
| < STRING  : "string" >
| < FLOAT : "float" >
| < CHAR : "char" >

}
```

```
  <CHAR_LITERAL: "'"  (~["\n","\r"])  "'">
|
  <FLOAT_LITERAL:[ "1"-"9" ] ([ "0"-"9" ])* (".") (["0"-"9"])+ >
|
```

```
Valor PValorFloat() :
{
    Token token;
}
{
    token = <FLOAT_LITERAL>
    {
        return new ValorFloat(Float.parseFloat(token.toString()));
    }
}
```

```
Valor PValorChar() :
{
    Token token;
}
{
    token = <CHAR_LITERAL>
    {
        String tokenStr = token.toString();
        tokenStr = tokenStr.substring(1,tokenStr.length()-1);
        char[] thetoken = tokenStr.toCharArray();
        return new ValorChar(thetoken[0]);
    }
}
```

Parser - LOOKAHEAD

```
Expressao PExpBinaria() :
{
    Expressao retorno, param2;
}
{
    (
        LOOKAHEAD (PExpPrimaria() <CONCAT> )
          retorno = PExpConcat()
        | LOOKAHEAD (PExpPrimaria() <MINUS>)
          retorno = PExpSub()
        | LOOKAHEAD (PExpPrimaria() <AND>)
          retorno = PExpAnd()
        | LOOKAHEAD (PExpPrimaria() <OR>)
          retorno = PExpOr()
        | LOOKAHEAD (PExpPrimaria() <EQ>)
          retorno = PExpEquals()
        | LOOKAHEAD (PExpPrimaria() <PLUS>)
          retorno = PExpSoma()
        | LOOKAHEAD (PExpPrimaria() <STAR>)
          retorno = PExpMult()
        | LOOKAHEAD (PExpPrimaria() <SLASH>)
          retorno = PExpDiv()
    )
    {
        return retorno;
    }
}
```

```
ExpMult PExpMult() :
{
    Expressao esq;
    Expressao dir;
}
{
    esq = PExpPrimaria()
    <STAR>
    dir = PExpressao()
    {return new ExpMult(esq, dir);}
}

ExpDiv PExpDiv() :
{
    Expressao esq;
    Expressao dir;
}
{
    esq = PExpPrimaria()
    <SLASH>
    dir = PExpressao()
    {return new ExpDiv(esq, dir);}
}
```

Vinculação de tipos dinâmica

“A vinculação de tipos dinâmica permite valores de quaisquer tipos serem atribuídos a quaisquer variáveis. Tipos incorretos de lados direitos de atribuições não são detectados como erros; em vez disso, o tipo do lado esquerdo é trocado para o tipo incorreto.”

Robert Sebesta,

Conceitos de Linguagem de Programação (9ª Edição)

Atribuição

```
public boolean checaTipo(AmbienteCompilacaoImperativa ambiente)
        throws VariavelNaoDeclaradaException, VariavelJaDeclaradaException {
        ambiente.map(id,expressao.getTipo(ambiente));

        return expressao.checaTipo(ambiente)
            && id.getTipo(ambiente).eIgual(expressao.getTipo(ambiente));
}
```

Sobrecarga nos Construtores - Int

```java
public ValorInteiro(Integer valor) {
    super(valor);
}

public ValorInteiro(Boolean valor){
    super(BoolMap(valor));
}

public ValorInteiro(Character valor){
    super((int)valor);
}
```

```java
private static Integer BoolMap(boolean valor){
    if(valor){
        return 1;
    }
    else{
        return 0;
    }
}
```

Sobrecarga nos Construtores - Float e String

```java
public ValorFloat(Float valor) {
    super(valor);
}

public ValorFloat(Integer valor){
    super((float) valor);
}
```

```java
public ValorString(String valor) {
    super(valor);
}

public ValorString(Integer valor){
    super(Integer.toString(valor));
}

public ValorString(Float valor){
    super(Float.toString(valor));
}

public ValorString(Boolean valor){
    super(Boolean.toString(valor));
}

public ValorString(Character valor){
    super(Character.toString(valor));
}
```

Soma

```
if(getEsq().avaliar(amb) instanceof ValorFloat && getDir().avaliar(amb) instanceof ValorInteiro){
    return new ValorFloat(((ValorFloat) getEsq().avaliar(amb)).valor() +
    ((ValorInteiro) getDir().avaliar(amb)).valor());
}
if(getEsq().avaliar(amb) instanceof ValorInteiro && getDir().avaliar(amb) instanceof ValorInteiro){
    return new ValorInteiro(((ValorInteiro) getEsq().avaliar(amb)).valor() +
    ((ValorInteiro) getDir().avaliar(amb)).valor());
}
if(getEsq().avaliar(amb) instanceof ValorInteiro && getDir().avaliar(amb) instanceof ValorFloat){
    return new ValorFloat(((ValorInteiro) getEsq().avaliar(amb)).valor() +
    ((ValorFloat) getDir().avaliar(amb)).valor());
}
```

Multiplicação

```
if(getEsq().avaliar(amb) instanceof ValorBooleano && getDir().avaliar(amb) instanceof ValorBooleano){
    ValorInteiro aux_1 = new ValorInteiro(((ValorBooleano) getDir().avaliar(amb)).valor());
    ValorInteiro aux_2 = new ValorInteiro(((ValorBooleano) getEsq().avaliar(amb)).valor());
    return new ValorInteiro(aux_1.valor() * aux_2.valor());
}
if(getEsq().avaliar(amb) instanceof ValorBooleano && getDir().avaliar(amb) instanceof ValorInteiro){
    ValorInteiro aux_1 = new ValorInteiro(((ValorInteiro) getDir().avaliar(amb)).valor());
    ValorInteiro aux_2 = new ValorInteiro(((ValorBooleano) getEsq().avaliar(amb)).valor());
    return new ValorInteiro(aux_1.valor() * aux_2.valor());
}
if(getEsq().avaliar(amb) instanceof ValorFloat && getDir().avaliar(amb) instanceof ValorBooleano){
    ValorInteiro aux_1 = new ValorInteiro(((ValorBooleano) getDir().avaliar(amb)).valor());
    return new ValorFloat(((ValorInteiro)aux_1).valor() * ((ValorFloat) getEsq().avaliar(amb)).valor());
}
```

Divisão

```
if(getEsq().avaliar(amb) instanceof ValorInteiro && getDir().avaliar(amb) instanceof ValorInteiro){
    return new ValorFloat(((ValorInteiro) getEsq().avaliar(amb)).valor() /
    ((ValorInteiro) getDir().avaliar(amb)).valor());
}
if(getEsq().avaliar(amb) instanceof ValorInteiro && getDir().avaliar(amb) instanceof ValorFloat){
    return new ValorFloat(((ValorInteiro) getEsq().avaliar(amb)).valor() /
    ((ValorFloat) getDir().avaliar(amb)).valor());
}
if(getEsq().avaliar(amb) instanceof ValorInteiro && getDir().avaliar(amb) instanceof ValorBooleano){
    ValorInteiro aux_1 = new ValorInteiro(((ValorBooleano) getDir().avaliar(amb)).valor());
    ValorInteiro aux_2 = new ValorInteiro(((ValorInteiro) getEsq().avaliar(amb)).valor());
    return new ValorInteiro(aux_2.valor() / aux_1.valor());
}
```

Concatenação

```
if(getEsq().avaliar(amb) instanceof ValorInteiro && getDir().avaliar(amb) instanceof ValorInteiro){
    ValorString aux_1 = new ValorString(((ValorInteiro) getDir().avaliar(amb)).valor());
    ValorString aux_2 = new ValorString(((ValorInteiro) getEsq().avaliar(amb)).valor());
    return new ValorString(aux_2.valor() + aux_1.valor());
}
if(getEsq().avaliar(amb) instanceof ValorBooleano && getDir().avaliar(amb) instanceof ValorInteiro){
    ValorString aux_1 = new ValorString(((ValorInteiro) getDir().avaliar(amb)).valor());
    ValorString aux_2 = new ValorString(((ValorBooleano) getEsq().avaliar(amb)).valor());
    return new ValorString(aux_2.valor() + aux_1.valor());
}
if(getEsq().avaliar(amb) instanceof ValorFloat && getDir().avaliar(amb) instanceof ValorBooleano){
    ValorString aux_1 = new ValorString(((ValorBooleano) getDir().avaliar(amb)).valor());
    ValorString aux_2 = new ValorString(((ValorFloat) getEsq().avaliar(amb)).valor());
    return new ValorString(aux_2.valor() + aux_1.valor());
}
```

Resumo

- Soma:
  - (Int|Bool|Char) + (Int|Bool|Char) = Int
  - Float + (Int|Bool|Char|Float) = Float
  - (Int|Bool|Char|Float) + Float = Float
- Multiplicação
  - (Int|Bool|Char) * (Int|Bool|Char) = Int
  - Float * (Int|Bool|Char|Float) = Float
  - (Int|Bool|Char|Float) * Float = Float

Resumo

- Divisão:
  - `(Char|Int|Bool) / Bool = Int`
  - `Float / (Float|Char|Int|Bool) = Float`
  - `(Float|Char|Int|Bool) / (Int|Char|Float) = Float`
- Concatenação:
  - `(Float|Char|Int|Bool|String) ++ (Float|Char|Int|Bool|String) = String`

Conclusão

- A coerção de tipos é uma parte importante da programação que permite a conversão automática de um tipo de dado em outro tipo de dado
- O programador deve estar ciente do processo de coerção para evitar erros inesperados