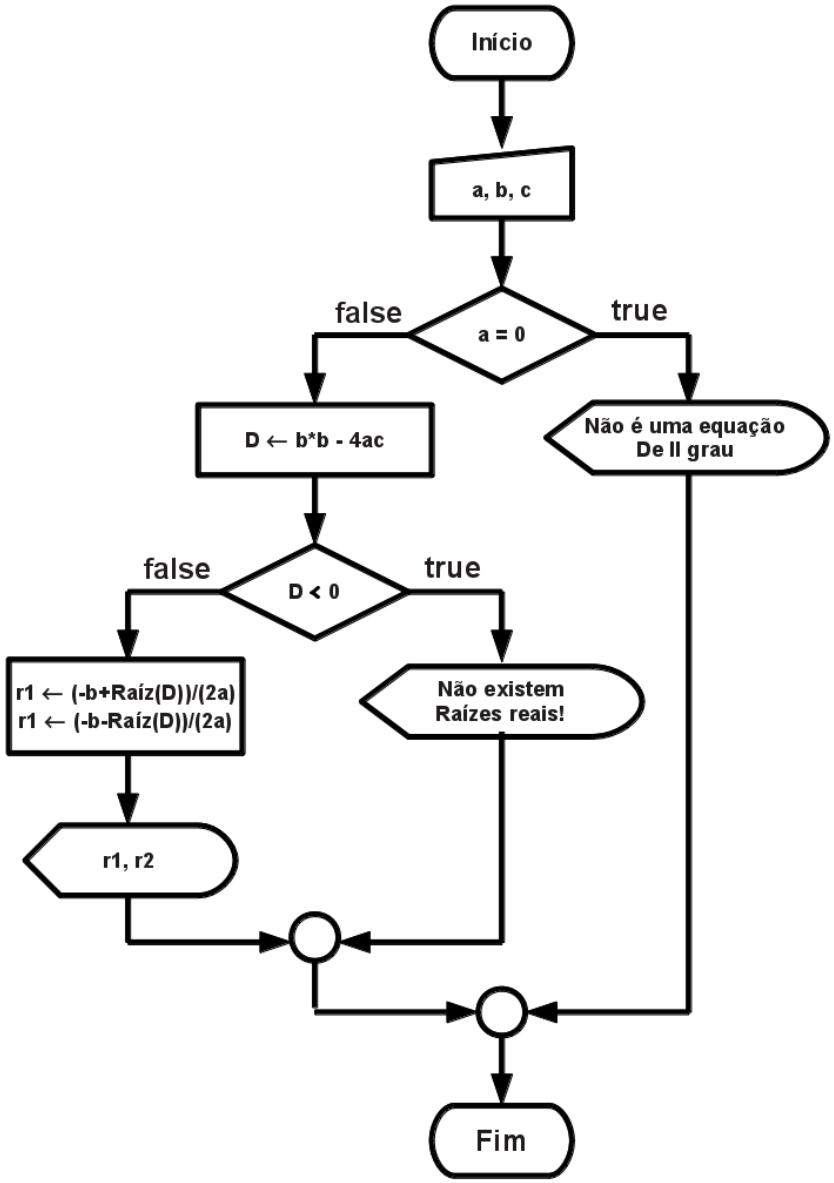
algoritmo

substantivo masculino

1. mat sequência finita de regras, raciocínios ou operações que, aplicada a um número finito de dados, permite solucionar classes semelhantes de problemas.
2. inf conjunto das regras e procedimentos lógicos perfeitamente definidos que levam à solução de um problema em um número finito de etapas.



Origem

ETIM lat.medv. algorismus, com infl. do gr. arithmós 'número'

Linguagem de Programação

Linguagem de programação é um método padronizado para comunicar instruções para um computador. É um conjunto de regras sintáticas e semânticas usadas para definir um programa de computador.

Linguagens de programação podem ser usadas para expressar algoritmos com precisão.

```
#include <math.h>
#include <stdio.h>

main()
{
  float a, b, c, d, r1, r2;

  printf("\nDigite o valor de A: ");
  scanf("%f", &a);
  printf("\nDigite o valor de B: ");
  scanf("%f", &b);
  printf("\nDigite o valor de C : ");
  scanf("%f", &c);

  if ( a == 0 )
    printf("Nao e uma equacao de II
grau\n");
  else
  {
    d = (b*b) - (4*a*c);
    if(d < 0)
      printf("Nao existem raizes reais\n");
    else
    {
      r1 = (-b + sqrt(d))/(2*a);
      r2 = (-b - sqrt(d))/(2*a);
      printf("A raiz x1 = %f\n", r1);
      printf("A raiz x2 = %f\n", r2);
    }
    return(0);
  }
}
```

Linguagem de Programação

A primeira máquina programável foi o tear programável de Jacquard que utilizava cartões perfurados para representar os padrões que deveriam ser tecidos.

Ada Lovelace foi a primeira pessoa a definir um algoritmo para resolver um problema em uma máquina programável. Apesar da Máquina Analítica de Charles Babbage não ter sido concluída, esse é considerado o primeiro programa de computador da história.

Linguagem de Máquina

Os processadores são capazes de executar apenas programas em sua própria linguagem binária, chamada linguagem de máquina. As linguagens de máquina são as primeiras linguagens de programação dos computadores. Devido a grande dificuldade para programar em código binário, foram criadas outras linguagens de programação para facilitar o desenvolvimento de programas.

Address	Machine Language			
0000 0000	0000	0000	0000	0000
0000 0001	0000	0000	0000	0010
0000 0010	0000	0000	0000	0011
0000 0011	0001	1101	0000	0001
0000 0100	0001	1110	0000	0010
0000 0101	0101	1111	1101	1110
0000 0110	0010	1111	0000	0000
0000 0111	1111	0000	0000	0000

Assembly

Assembly ou linguagem de montagem é uma notação legível por humanos para o código de máquina que uma arquitetura de computador específica usa. A linguagem de máquina torna-se legível pela substituição dos valores binários por símbolos chamados mnemônicos As linguagens de montagens foram as primeiras linguagens de programação legíveis.

Address	Machine Language				Assembly Language
0000 0000	0000	0000	0000	0000	TOTAL .BLOCK 1
0000 0001	0000	0000	0000	0010	ABC .WORD 2
0000 0010	0000	0000	0000	0011	XYZ .WORD 3
0000 0011	0001	1101	0000	0001	LOAD REGD, ABC
0000 0100	0001	1110	0000	0010	LOAD REGE, XYZ
0000 0101	0101	1111	1101	1110	ADD REGF, REGD, REGE
0000 0110	0010	1111	0000	0000	STORE REGF, TOTAL
0000 0111	1111	0000	0000	0000	HALT

Nível de Abstração

Apesar de legível, as linguagens de montagem são voltadas para a arquitetura do computador e são de difícil utilização. Para facilitar a programação, foram criadas linguagens de programação voltadas para a linguagem do programador. Com relação ao grau de abstração em relação a arquitetura do computador, as linguagens são divididas em:

Linguagem de baixo nível: linguagem que compreende as características da arquitetura do computador. utiliza somente instruções do processador (linguagem de máquina e assembly).

Linguagem de alto nível: são linguagens com um nível de abstração relativamente elevado, mais próximas da linguagem humana, da forma como são descritos os algoritmos, por exemplo: Fortran, Java, Php.

Linguagem de nível médio: linguagens de alto nível que permitem acesso a recursos de baixo nível como registradores e endereços de memória e inclusão de código de baixo nível, podem ser consideradas como linguagens de nível médio, embora essa classificação não seja frequentemente usada.

Código Fonte

Com a utilização de linguagens de alto nível passamos a ter uma grande diferença entre o programa executável pelo computador em linguagem de máquina e o programa escrito na linguagem de alto nível que passa a ser definido como Código Fonte. O Projeto de Informação do Linux define código fonte como "*uma versão do software da forma em que ele foi originalmente escrito (digitado em um computador) por um humano em texto puro (caracteres alfanuméricos humanamente legíveis)*". A noção de código fonte também pode ser aplicada de maneira mais abrangente, incluindo notações em linguagens gráficas.

Biblioteca

Nas linguagens de alto nível é possível deixar pré-definidas várias funções e outras funcionalidades que são utilizadas no desenvolvimento dos programas, evitando que o programador tenha que criar essas funcionalidades novamente para cada programa. Esses trechos de código pré-definidos são agrupados em bibliotecas que são uma coleção de subprogramas já compilados, que são adicionados ao programa posteriormente ao processo de desenvolvimento do código fonte. Existem bibliotecas disponibilizadas junto com as ferramentas da linguagem, que fazem parte da própria definição da linguagem. Além dessas bibliotecas, o desenvolvedor pode criar as próprias bibliotecas para facilitar o uso de código comum a vários programas.

Formas de Execução

Como os processadores executam apenas programas em sua própria linguagem de máquina, é necessário traduzir os programas em outras linguagens para a linguagem de máquina do computador onde o programa será executado.

Códigos escritos em Assembly são traduzidos por um programa chamado montador ou assembler que traduz as instruções da linguagem de montagem para código de máquina. Além dos comandos básicos, que são traduzidos diretamente para a linguagem de máquina, alguns montadores também aceitam diretivas, que são comandos específicos para o montador. Por exemplo, é possível definir constantes na memória utilizando diretivas.

Formas de Execução

Programas em linguagens de alto nível podem ser traduzidos por compiladores para serem transformados em programas em linguagem de máquina que pode ser executado diretamente pelo sistema operacional do computador ou executados por um programa intermediário que interpreta o código fonte cada vez que executa o programa.

Compilação

Para linguagens compiladas, o código fonte do programa é transformado passando por um programa chamado compilador e depois por outro programa chamado linker, gerando um programa executável que pode ser carregado na memória e executado pelo sistema operacional.

A compilação de um programa tem as seguintes fases:

- Pré-processamento
- Análise Léxica
- Análise Sintática
- Análise Semântica
- Geração do Código Objeto

Pré-Processamento

Algumas linguagens possuem um estágio antes da compilação onde um pré-processador realiza alterações no código fonte como a substituição de constantes, substituição de macros e inclusão de outros arquivos. Nem todas as linguagens tem um pré-processamento antes da compilação.

Por exemplo, o código:

```
#define max(a,b) (a<b ? b : a)
#define TAM 3

main() {
  int i[TAM],j,k;
  i[0] = max(j,k);
}
```

É transformado em:

```
main() {
  int i[3];
  i[0] = (j<k ? k : j);
}
```

Análise Léxica

A função do analisador léxico é ler o código fonte, buscando a separação e identificação dos elementos componentes do programa fonte, denominados símbolos léxicos ou tokens, é também de responsável pela eliminação de elementos "decorativos" do programa, tais como espaços em branco, marcas de formatação de texto e comentários.

Análise Sintática

A análise sintática, ou análise gramatical é o processo de se determinar se uma cadeia de símbolos léxicos pode ser gerada por uma gramática. O analisador sintático é o cerne do compilador, responsável por verificar se os símbolos contidos no programa fonte formam um programa válido, ou não.

Análise Semântica

O papel do analisador semântico é prover métodos pelos quais as estruturas construídas pelo analisador sintático possam ser avaliadas ou executadas. É papel do analisador semântico assegurar que todas as regras sensíveis ao contexto da linguagem estejam analisadas e verificadas quanto à sua validade. Um exemplo de tarefa própria do analisador semântico é a checagem de tipos de variáveis em expressões.

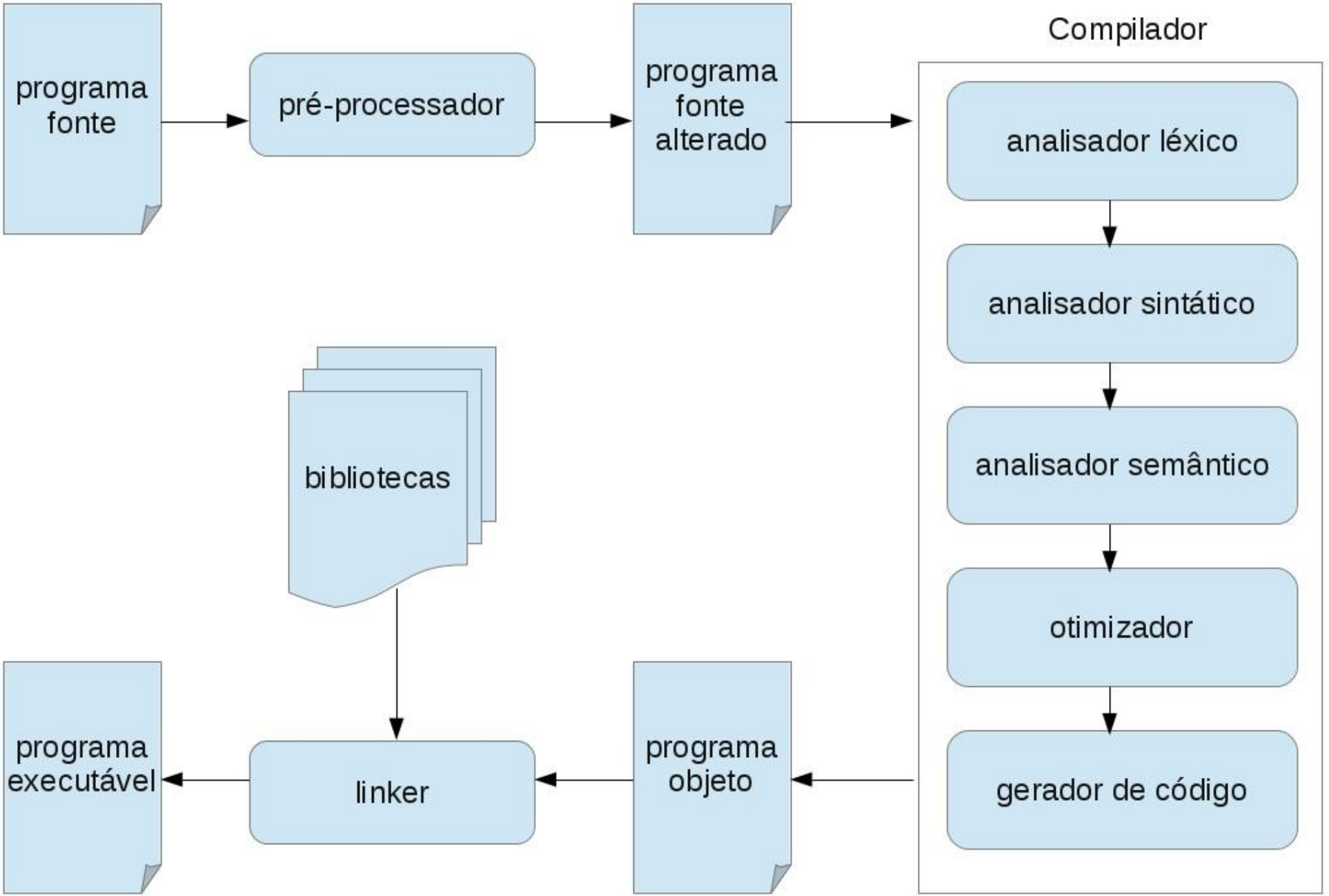
Geração do Código Objeto

O compilador gera um código objeto que contém o código em linguagem de máquina para executar as instruções do código fonte, informações sobre alocação de memória, os símbolos do programa (como nomes de variáveis e de funções) e também informações sobre debug. Porém o código em linguagem de máquina corresponde a funções e objetos que pertencem a bibliotecas da linguagem não são colocados no código objeto e devem ser ligados ao código objeto por um programa de ligação ou linker.

Alguns compiladores executam uma otimização do programa antes de gerar o código objeto.

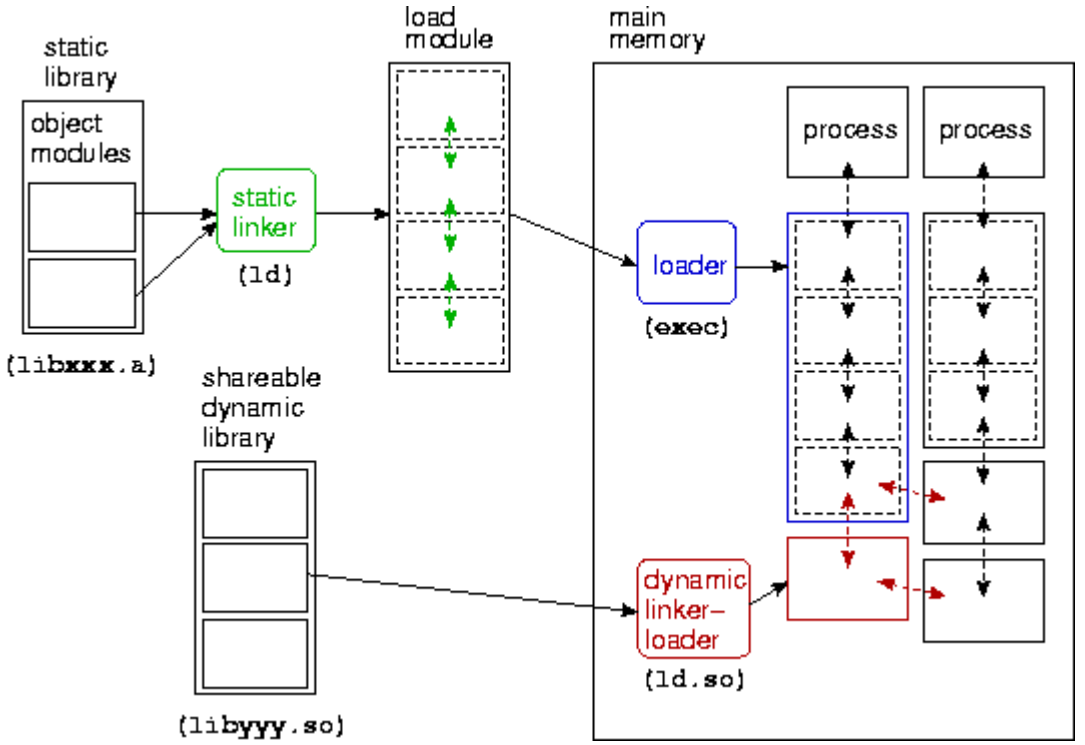
Linkedição

Para que o programa objeto possa ser executado, é necessário acrescentar a esse programa o código referente a funções e objetos das bibliotecas da linguagem. Esse código pode ser ligado estaticamente por um programa chamado linker, gerando o programa executável que contém o código de máquina correspondente ao código fonte e as bibliotecas da linguagem.


programa fonte
pré-processador
programa fonte alterado
Compilador
analisador léxico
analisador sintático
analisador semântico
otimizador
gerador de código
bibliotecas
programa executável
linker
programa objeto

Loader

Para executar um programa, o loader do sistema operacional carrega esse programa na memória e executa as tarefas necessárias para iniciar a execução do programa. Uma das tarefas é ligar ao programa executável o código das funções de bibliotecas dinâmicas, que não são incluídos no executável pelo linker.

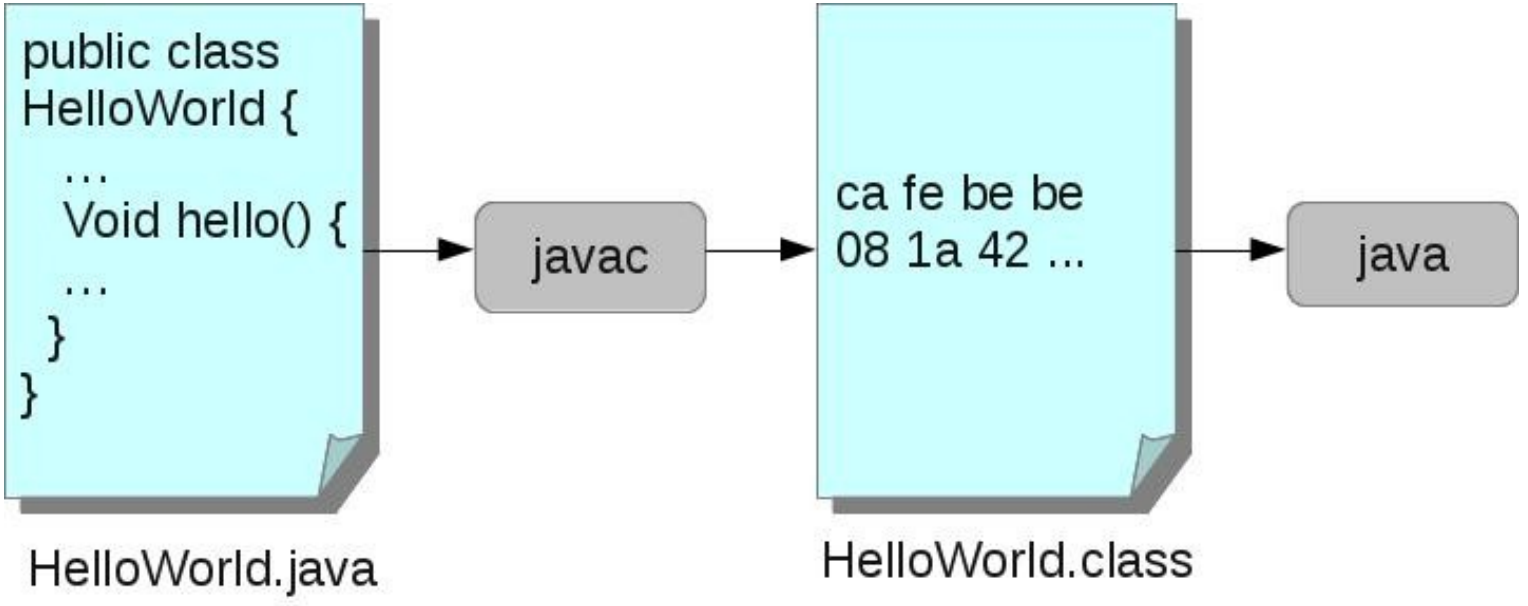


Interpretação

Programas em linguagens interpretadas não são executados diretamente pelo sistema operacional do computador. O código fonte na linguagem interpretada é lido por um programa (um interpretador ou uma máquina virtual) que faz a análise do código fonte e executa as operações descritas no código fonte. Em programas puramente interpretados, cada linha de código é analisada e interpretada cada vez que é executada. Dessa forma, a execução de programas interpretados é mais lenta do que de programas compilados.

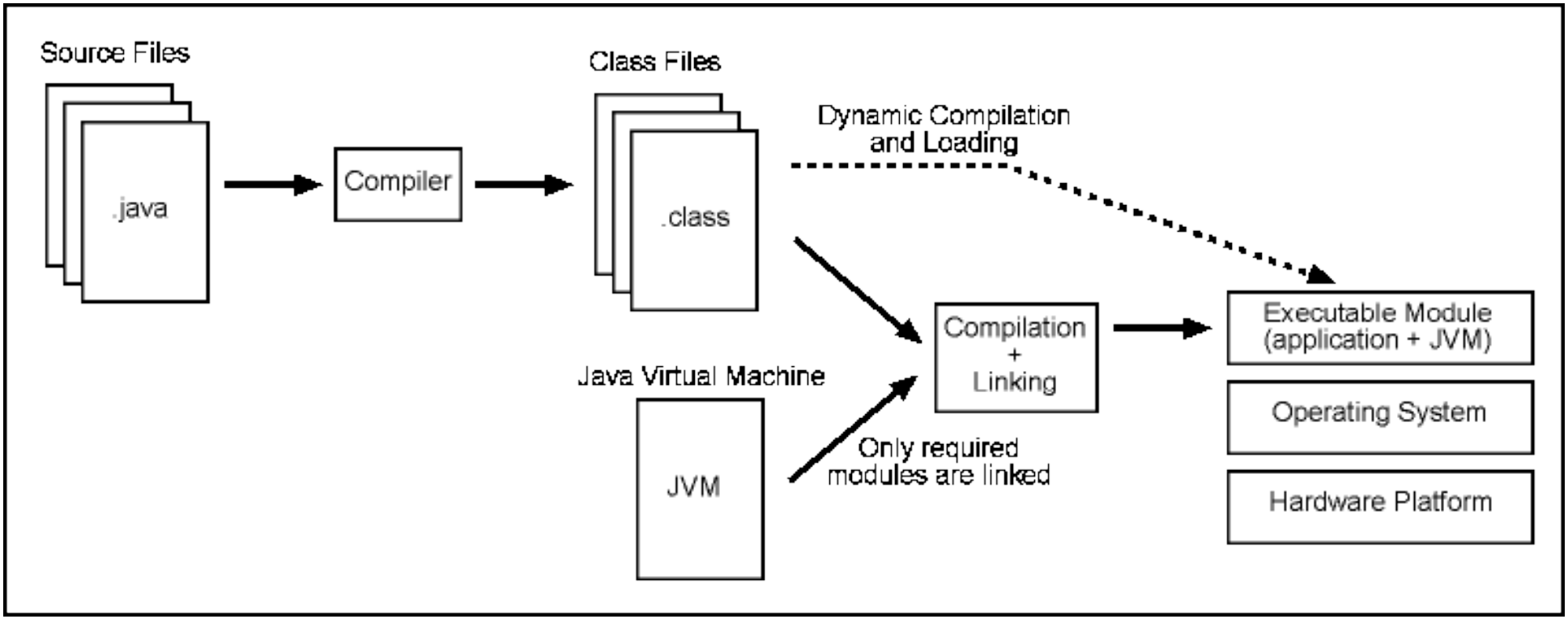
Interpretação

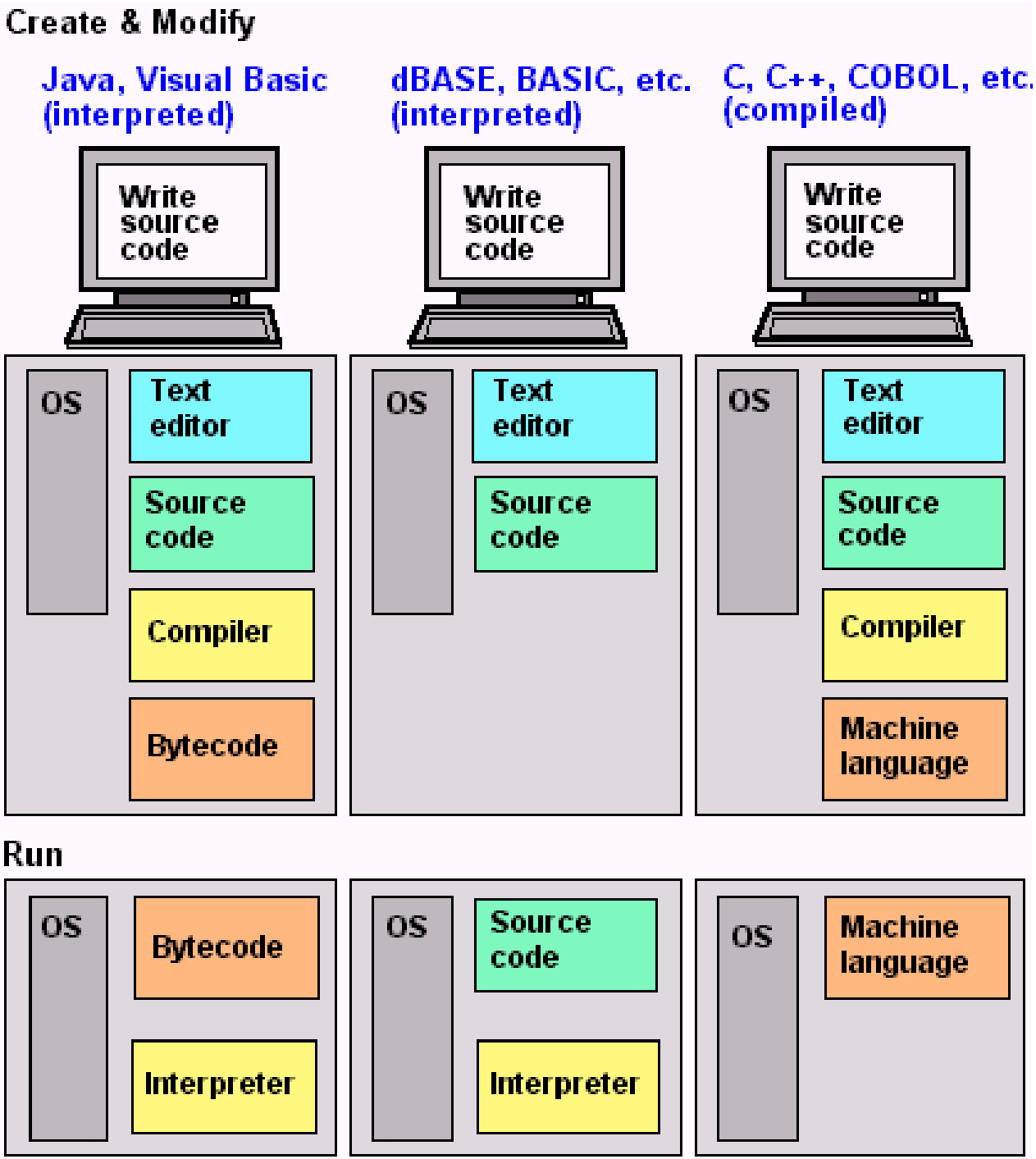
Algumas linguagens interpretadas podem primeiro ser compiladas para bytecodes, gerando um programa intermediário que é executado por uma máquina virtual. O programa em bytecode não pode ser executado diretamente pelo processador, mas não precisa passar por todos os processos de interpretação cada vez que é executado.



Interpretação

Outra forma de acelerar a execução de linguagens interpretadas é utilizando compiladores que geram um programa executável que incluem um programa intermediário em bytecode e a máquina virtual em um único arquivo.




Create & Modify
Java, Visual Basic (interpreted)
dBASE, BASIC, etc. (interpreted)
C, C++, COBOL, etc. (compiled)
Write source code
Write source code
Write source code
OS
Text editor
Source code
Compiler
Bytecode
OS
Text editor
Source code
OS
Text editor
Source code
Compiler
Machine language
Run
OS
Bytecode
Interpreter
OS
Source code
Interpreter
OS
Machine language

Primórdios

Plankalkül foi a primeira linguagem de programação de alto nível, criada por Konrad Zuse nos anos 40 mas só foi implementada em 1998.

Short Code, proposta por John Mauchly em 1949, foi a primeira linguagem de alto nível implementada para computadores eletrônicos, o programa tinha que ser traduzido para linguagem de máquina cada vez que executado, tornando a execução do programa muito mais lento do que programas nativos em linguagem de máquina.

Considera-se que Grace Hopper criou o primeiro compilador em 1952 para a linguagem A-0.

Primórdios

As primeiras linguagens modernas foram:

- FORTRAN (1954), "FORmula TRANslator", inventada por John Backus e outros.;
- LISP (1958), "LISt Processor", inventada por John McCarthy e outros.;
- COBOL (1959), “COmmon Business Oriented Language”, criada pelo Short Range Committee, com grande influência de Grace Hopper.

Outra família de linguagens importante da época foi ALGOL (1958), “ALGOrithmic Language”, que introduziu os elementos das linguagens estruturadas.

Fortran

Fortran é uma família de linguagens desenvolvida a partir da década de 1950. Uma das linguagens mais usadas em aplicações científicas, ainda é utilizada pelo meio científico. É considerada uma das primeiras linguagens de programação imperativas. Apesar de ter sido inicialmente uma linguagem de programação procedural, versões recentes de Fortran possuem características que permitem suportar programação orientada por objetos.

```
C 1 2 3 4 5 6
C234567890123456789012345678901234567890123456789012345678901234567890123456789012345
      PROGRAM BASKHARA
C
      REAL :: A,B,C, DELTA, X1,X2, RE, IM
      PRINT *, "Este programa resolve uma equação de 2o.grau"
      PRINT *, "no formato: a*x**2 + b*x + c = 0"
      PRINT  10, "Digite a, b, c: "
 10   FORMAT( A, 1X, $)
 20   READ(*, *, ERR=20)A, B, C
C
      DELTA= B**2-4.0*A*C
      IF( DELTA.GT.0 )THEN      ! (DUAS RAÍZES REAIS)
          X1= ( -B-SQRT(DELTA) ) / ( 2.0*A )
          X2= ( -B+SQRT(DELTA) ) / ( 2.0*A )
          PRINT *, "RAIZES:  X1= ", X1
          PRINT *, "X2= ", X2
      ELSE
          IF( DELTA.EQ.0 ) THEN ! (DUAS RAÍZES REAIS IGUAIS)
          X1= -B / ( 2.0*A )
          X2= X1
          PRINT *, "RAIZES: X1=X2= ", X1
      ELSE                      ! (DUAS RAÍZES COMPLEXAS)
          RE= -B / ( 2.0*A )
          IM= SQRT( -DELTA ) / ( 2.0*A )
          PRINT *, "RAIZES COMPLEXAS: X1= ", RE,"- ", IM, "i"
          PRINT *, "X2= ", RE, "+ ",IM, "i"
          END IF
      END IF
      END PROGRAM BASKHARA
```

Lisp

Lisp é uma família de linguagens de programação concebida por John McCarthy em 1958. É uma linguagem formal matemática baseada no processamento de listas. Utilizada na área de inteligência artificial, é uma linguagem funcional onde um programa é uma função (ou grupo de funções), tipicamente constituída de outras funções mais simples.

Exemplo:

```
(defun fatorial (n)
  (if (= n 0)
      1
      (* n (fatorial (- n 1)))))
```

COBOL

COBOL foi criado por um comitê de investigadores de várias instituições civis e governamentais durante o segundo semestre de 1959. Baseado nas linguagens FLOW-MATIC inventada pela Grace Hopper e na linguagem de programação da IBM COMTRAN. Voltada para o desenvolvimento de aplicações comerciais, foi uma das linguagens mais utilizadas no mundo. Ainda é muito utilizado em sistemas para Mainframes, principalmente sistemas legados. As especificações mais atuais incluem suporte à programação orientada a objetos.

```
001010 IDENTIFICATION DIVISION.
001020 PROGRAM-ID. TABSORT.
001030 AUTHOR. ALEX BASTOS.
001040 INSTALLATION. RDC-DIV USUARIOS.
001050 DATE-WRITEN. 18/12/72.
001060 DATE-COMPILED. 20/12/72.
001070 REMARKS. Este programa grava arquivo sequencial em ordem
       crescente e classifica-o apos em ordem decrescente.
001100 ENVIRONMENT DIVISION.
001110 CONFIGURATION SECTION.
001120 SOURCE-COMPUTER. IBM-370-165.
001130 OBJECT-COMPUTER. IBM-370-165.
001140 INPUT-OUTPUT SECTION
001150 FILE-CONTROL.
001160     SELECT ARQUIVO ASSIGN TO DA-S-DISCO.
001170     SELECT TRABALHO ASSIGN TO DA-S-SORTWK01.
001180 DATA DIVISION.
001190 FILE SECTION.
001200 FD ARQUIVO LABEL RECORDS ARE STANDARD
002010      DATA RECORD IS ENTRADA.
002020 01 ENTRADA.
002030      02 X PIC 99.
002040      02 Y PIC X(10).
002050      02 FILLER PIC X(20).
002060 SD TRABALHO LABEL RECORDS ARE STANDARD
002070      DATA RECORD IS TRAB.
002080 01 TRAB.
002090      02 Z PIC 99.
002100      02 K PIC X(10).
002110    02 FILLER PIC X(20).
002120 WORKING-STORAGE SECTION.
002130 77 I PIC 99 VALUE ZEROS.
```

```
002140 PROCEDURE DIVISION.
002150      OPEN OUTPUT ARQUIVO.
002160      MOVE 'TESTE-SORT' TO Y.
002170 GRAVACAO.
002180      MOVE I TO X.
002190      ADD 1 TO I.
002200      WRITE ENTRADA.
003010      IF I  100 GO TO GRAVACAO.
003020      SORT TRABALHO DESCENDING Z USING
003030      ARQUIVO GIVING ARQUIVO.
003040      OPEN INPUT ARQUIVO.
003050 GRAVA.
003060      READ ARQUIVO AT END GO TO FIN.
003070      DISPLAY X' Y.
003080      GO TO GRAVA.
003090 FIN.
003100 CLOSE ARQ.
003110 STOP RUN.
```

ALGOL

O ALGOL é uma família de linguagens de programação de alto nível voltadas principalmente para aplicações científicas. A definição do ALGOL 60 foi um evento-chave na história das linguagens de programação. Foi a primeira linguagem de programação estruturada e influenciou o desenvolvimento de um grande número de linguagens até nos dias atuais.

```
comment An Algol 60 sorting program;
procedure Sort (A, N)
  value N;
  integer N;
  real array A;
begin
  real X;
  integer i, j;
  for i := 2 until N do begin
    X := A[i];
    for j := i-1 step -1 until 1 do
      if X >= A[j] then begin
        A[j+1] := X; goto Found
      end else
        A[j+1] := A[j];
    A[1] := X;
  Found:
    end
  end
end Sort
```

Paradigmas de Programação

Um paradigma de programação fornece e determina a visão que o programador possui sobre a estruturação e execução do programa. Os paradigmas de programação são muitas vezes diferenciados pelas técnicas de programação que proíbem ou permitem. Podemos adotar várias divisões entre as linguagens segundo o paradigma adotado. Algumas divisões são:

Quanto a forma de execução:

- Programação Sequencial X Programação Concorrente

Quanto a forma de programação:

- Programação Imperativa X Programação Declarativa

Sequencial x Concorrente

Na programação sequencial as tarefas do programa são executadas sequencialmente, uma após a outra.

Na programação concorrente, as tarefas de um programa são executadas simultaneamente.

A execução simultânea das tarefas pode ocorrer de diversas formas e por diferentes objetivos. Dessa forma, a execução simultânea pode ser dividida em Programação Concorrente, Programação Paralela e Programação Distribuída.

Imperativa X Declarativa

A programação imperativa é um paradigma de programação que descreve a computação como ações, enunciados ou comandos que mudam o estado (variáveis) de um programa. O fundamento da programação imperativa é o conceito de Máquina de Turing.

A programação declarativa é um paradigma que expressa a lógica de uma computação sem descrever seu fluxo de controle. Descreve “o que” um programa deve fazer em vez de descrever “como fazer” com uma sequência de instruções explícitas.

Paradigmas Imperativos

Programação Procedural

Programação Procedural é às vezes utilizado como sinônimo de Programação Imperativa, mas pode se referir a um paradigma de programação baseado no conceito de chamadas a procedimento, também conhecidos como rotinas ou sub-rotinas As primeiras linguagens de programação procedural tinham como padrão o uso de GOTO e labels, o que tornava o código de difícil compreensão. Exemplos de linguagens procedurais:

- Fortran
- Cobol
- Basic

```
      program ex2

          implicit none
          integer a, b, v(4), i
          real media

10        read *, a, b
          if( a.EQ.0 ) then goto 20
          call troca(a, b)
          print *, a, b
          print *, media(a, b)
          read *, (v(i), i = 1, 4)
          call troca(v(1), v(4))
          call troca(v(2), v(3))
          print *, (v(i), i = 1, 4)
          goto 10
20        continue
      end
```

```
      subroutine troca(x, y)
          implicit none
          integer x, y, aux

          aux = x
          x = y
          y = aux
      end

      real function media(x, y)
          implicit none
          integer x, y

          media = (x + y) / 2.0
      end
```

Programação Estruturada

A Programação Estruturada preconiza que todos os programas possíveis podem ser reduzidos a apenas três estruturas: sequência, decisão e iteração (ou repetição). As linguagens estruturadas também são procedurais uma vez que se baseia na divisão do programa em estruturas simples, usando as sub-rotinas e as funções. Porém, ao invés do uso de GOTOs e labels, o programa deve ser estruturado em blocos com as estruturas de repetição. Exemplos de linguagens estruturadas:

- Algol 60
- C
- Pascal

```c
#include <stdio.h>

int main(int argc,char** argv)
{
  int x;
  while( 1 )
  {
    printf("Entre com um valor\n");
    scanf( "%d", &x);
    if ( x < 0 )
      break;
    int fat = 1;
    int i = 1;
    while( i <= x )
    {
      fat = fat * i;
      i++;
    }
    printf("Fatorial de %d e %d\n",x,fat);
  }
}
```

```
#include <stdlib.h>
#include <iostream>
using namespace std;

int area_quadrado(int lado)
{
  return lado * lado;
}

int perimetro_quadrado(int lado)
{
  return (4*lado);
}

int area_retangulo(int base, int
altura)
{
  return base * altura;
}

int perimetro_retangulo(int base,
int altura)
{
  return (2*base)+(2*altura);
}
```

```
int main(int argc,char** argv)
{
  int x = 5; int y = 3;
  int z = 6;
  cout << "Retangulo" << endl;
  cout << "Area      = "
   << area_retangulo(x,y)
   << endl;
  cout << "Perimetro = "
   << perimetro_retangulo(x,y)
   << endl;
  cout << "Quadrado" << endl;
  cout << "Area      = "
   << area_quadrado(z)
   << endl;
  cout << "Perimetro = "
   << perimetro_quadrado(z)
   << endl;
  return (EXIT_SUCCESS);
}
```

Programação Orientada a Objetos

A orientação a objetos é um modelo de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos. Implementa-se um conjunto de classes que definem os objetos presentes no sistema de software. Cada classe determina o comportamento (métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos. Exemplos de linguagens orientadas a objetos:

- C++
- Java
- Ruby

```
#include <stdlib.h>
#include <iostream>
using namespace std;

class quadrado {
  public:
  int lado;

  int area() {
    return lado * lado; }

  int perimetro() {
    return (4*lado); }

  void printdata() {
    cout << "Area      = "
      << area() << endl;
    cout << "Perimetro = "
      << perimetro() << endl; }
};

class retangulo {
  public:
  int bas, alt;

  int area() {
    return bas * alt; }

  int perimetro() {
    return (2*bas)+(2*alt); }

  void printdata() {
    cout << "Area      = "
      << area() << endl;
    cout << "Perimetro = "
      << perimetro() << endl; }
};

int main(int argc,char** argv)
{
  retangulo x;
  quadrado y;
  x.bas = 5; x.alt = 3;
  y.lado = 6;
  cout << "Retangulo" << endl;
  x.printdata();
  cout << "Quadrado" << endl;
  y.printdata();
  return (EXIT_SUCCESS);
}
```

Programação Orientada a Eventos

A Programação Orientada a Eventos é um paradigma onde o fluxo do programa é controlado por eventos como ações do usuário(clique no mouse, pressionar de teclas), e mensagens de outros programas. É voltada para a interface com o usuário. Geralmente a aplicação tem um laço de repetição de eventos, que recebem repetidamente informação para processar e disparam uma função de resposta de acordo com o evento. A orientação a eventos pode ser implementada em qualquer linguagem de programação mas é mais fácil em linguagens que já implementam mecanismos para tratamento de eventos com java e c#.

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class Evento implements ActionListener {

  public static void main(String[] args) {
    Evento instance = new Evento();
    JFrame frame = new JFrame("Test");
    frame.setLayout(new FlowLayout());
    frame.setSize(200, 100);

    JButton hello = new JButton("Hello");
    hello.setActionCommand("Hello");
    hello.addActionListener(instance);
    frame.add(hello);

    JButton goodbye = new JButton("Goodbye");
    goodbye.setActionCommand("Goodbye");
    goodbye.addActionListener(instance);
    frame.add(goodbye);

    frame.setVisible(true);
  }
```

```
  @Override
  public void actionPerformed(ActionEvent evt) {
    if (evt.getActionCommand() == "Hello") {
      JOptionPane.showMessageDialog(null, "Hello");
    } else if (evt.getActionCommand() == "Goodbye") {
      JOptionPane.showMessageDialog(null, "Goodbye");
    }
  }
}
```

Paradigmas Declarativos

Programação Funcional

Programação Funcional é um paradigma de programação que trata a computação como uma avaliação de funções matemáticas e que evita estados ou dados mutáveis. Ela enfatiza a aplicação de funções, em contraste da programação imperativa, que enfatiza mudanças no estado do programa. Uma função, neste sentido, pode ter ou não ter parâmetros e um simples valor de retorno. Os parâmetros são os valores de entrada da função, e o valor de retorno é o resultado da função. A definição de uma função descreve como a função será avaliada em termos de outras funções. Linguagens funcionais são usadas mais amplamente em ambiente acadêmico, mas algumas linguagens também tem sido usadas na indústria e no comércio. Exemplos de linguagens funcionais são Lisp, Scheme, Erlang, Mathematica.

Programação Lógica

Programação Lógica é um paradigma de programação que faz uso da lógica matemática para programação. Nesse paradigma, um programa é um conjunto de sentenças lógicas, expressando fatos e regras sobre o domínio do problema, permitindo responder questões sobre o problema. A principal aplicação desse paradigma é na área de inteligência artificial. Exemplos:

- Planner
- Prolog
- Datalog

Em um exemplo em Prolog teriamos:

Fatos:

```
man(adam).
man(peter).
man(paul).
woman(marry).
woman(eve).
```

Relações:

```
parent(adam,peter). % means adam is parent of peter
parent(eve,peter).
parent(adam,paul).
parent(marry,paul).
```

Regras:

```
father(F,C):-man(F),parent(F,C).
mother(M,C):-woman(M),parent(M,C).
```

Isso possibilita responder questões como:

```
?-father(X,paul).
```

Cuja resposta é X=adam

Programação Restritiva

Programação com restrições é um paradigma de programação que se refere ao uso de restrições na construção de relações entre variáveis. Consiste em especificar, para uma solução, que critérios (restrições) esta tem de cumprir. Surgiu inicialmente contido no contexto da programação lógica, apesar de atualmente existirem implementações baseadas em programação funcional (como em Oz) e programação imperativa (como em Kaleidoscope).

Estrutura de Tipos

Com relação a forma como tratam os tipos de dados, as linguagens podem ser divididas em:

Quanto a tipagem das variáveis:

- Estaticamente Tipadas X Dinamicamente Tipadas

Quanto a conversão de tipos:

- Fortemente Tipadas X Fracamente Tipadas

É frequente que os conceitos estaticamente tipadas e fortemente tipadas e os conceitos dinamicamente tipadas e fracamente tipadas sejam considerados como semelhantes, mas isso não e adequado.

Estaticamente Tipadas X Dinamicamente Tipadas

Nas linguagens estaticamente tipadas o tipo de dados de uma variável é definido em tempo de compilação e a variável não pode mudar de tipo durante a execução do programa. É obrigatória a declaração da variável. Linguagens estaticamente tipadas permitem a verificação de tipos pelo compilador, o que facilita a criação de programas mais seguros. Exemplos: java, c/c++, cobol, fortran.

Nas linguagens dinamicamente tipadas o tipo de uma variável é definido em tempo de execução e geralmente a variável pode mudar de tipo durante a execução do programa. Geralmente não exigem a declaração de variáveis. Mais flexíveis porém com maior possibilidade de erros de programação. Mais comuns em linguagens interpretadas. Exemplos: perl, php, python, ruby.

Fortemente Tipadas X Fracamente Tipadas

Nas linguagens fortemente tipadas há regras rígidas para a conversão de tipos, o uso de tipos de dados diferentes em atribuições e expressões exige a conversão explícita dos tipos se não houver uma regra definida para a conversão. Mais comum em linguagens compiladas e estaticamente tipadas (mas não obrigatoriamente ou exclusivamente), a exigência de conversão explícita diminui a chance de erros de programação. Exemplos: c++, ruby, pascal, python.

Nas linguagens fracamente tipadas a conversão de tipos é feita implicitamente, permitindo que atriuições e expressões com tipos de dados diferentes sejam usados mais livremente sem a necessidade de uma conversão explícita pelo programador. Mais comum em linguagens interpretadas e dinamicamente tipadas (mas não obrigatoriamente ou exclusivamente) também são mais flexíveis porém com maior possibilidade de erros de programação. Exemplos: php, c, perl.

Gerações

Existem diferentes classificações das gerações das linguagens de programação sendo mais comum a seguinte divisão:

- 1ª geração: linguagens em nível de máquina
- 2ª geração: linguagens de montagem (Assembly)
- 3ª geração: linguagens orientadas ao usuário
- 4ª geração: linguagens orientadas à aplicação
- 5ª geração: linguagens de conhecimento

Porém a classificação por geração apresenta diversas inadequações como incluir linguagens que não são linguagens de programação como SQL, confundir conceitos de linguagem de programação com ambiente de desenvolvimento e a impressão de que linguagens voltadas a inteligência artificial são uma evolução (nova geração) em relação a linguagens imperativas, quando na verdade, essas linguagens existem praticamente desde o início da história da programação, não sendo então uma nova geração mas uma linha diferente, voltada para outras formas de aplicação.

IDE

IDE (Integrated Development Environment) é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo. Uma IDE possui um conjunto de ferramentas para auxiliar o desenvovimento da aplicação, sendo as mais comuns:

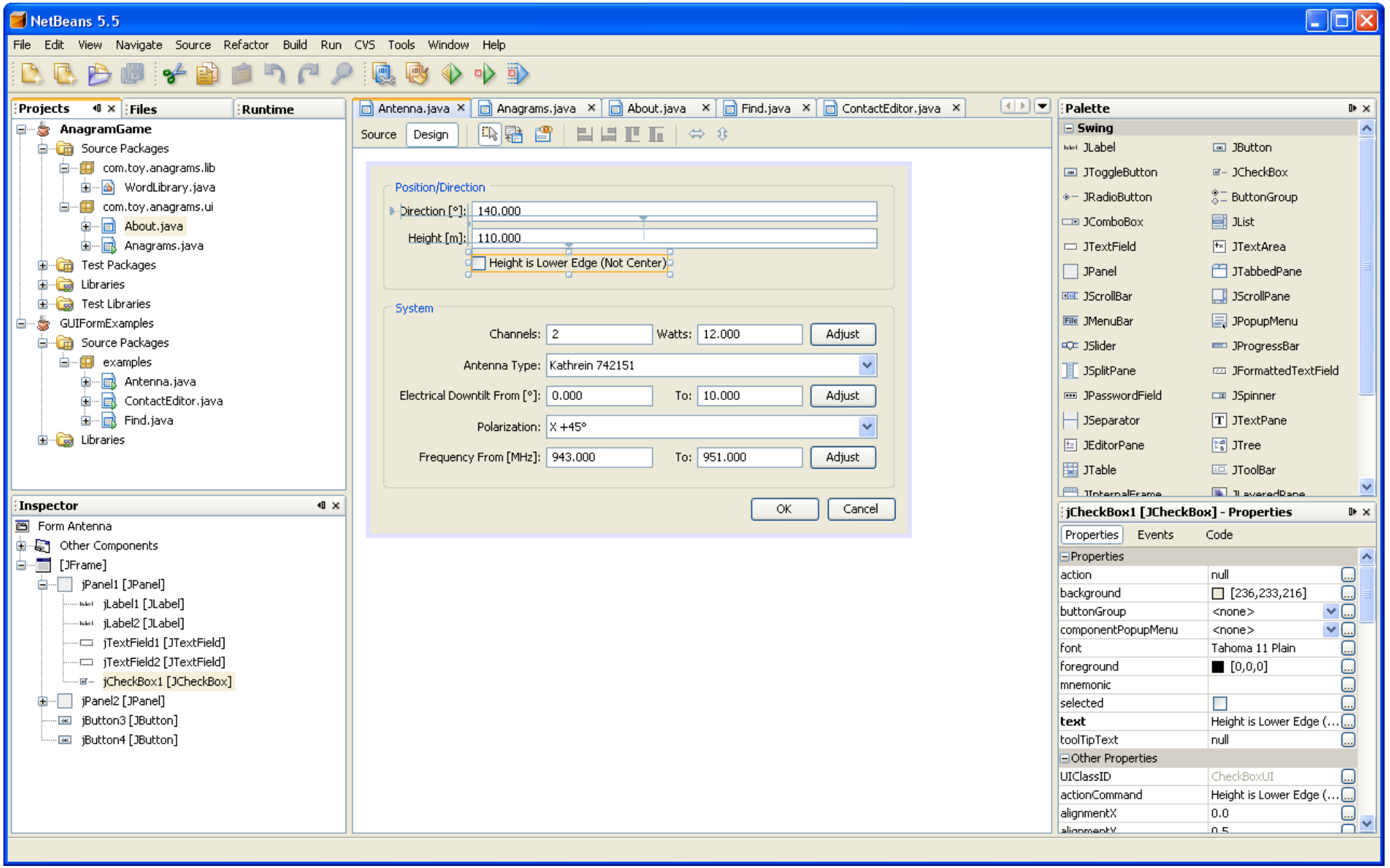
- Editor - edita o código-fonte do programa escrito na(s) linguagem(ns) suportada(s) pela IDE
- Compilador - compila o código-fonte do programa
- Linker - liga os vários "pedaços" de código-fonte, compilados em linguagem de máquina, em um programa executável
- Debugger - auxilia no processo de encontrar e corrigir defeitos no código-fonte do programa, na tentativa de aprimorar a qualidade de software

IDE

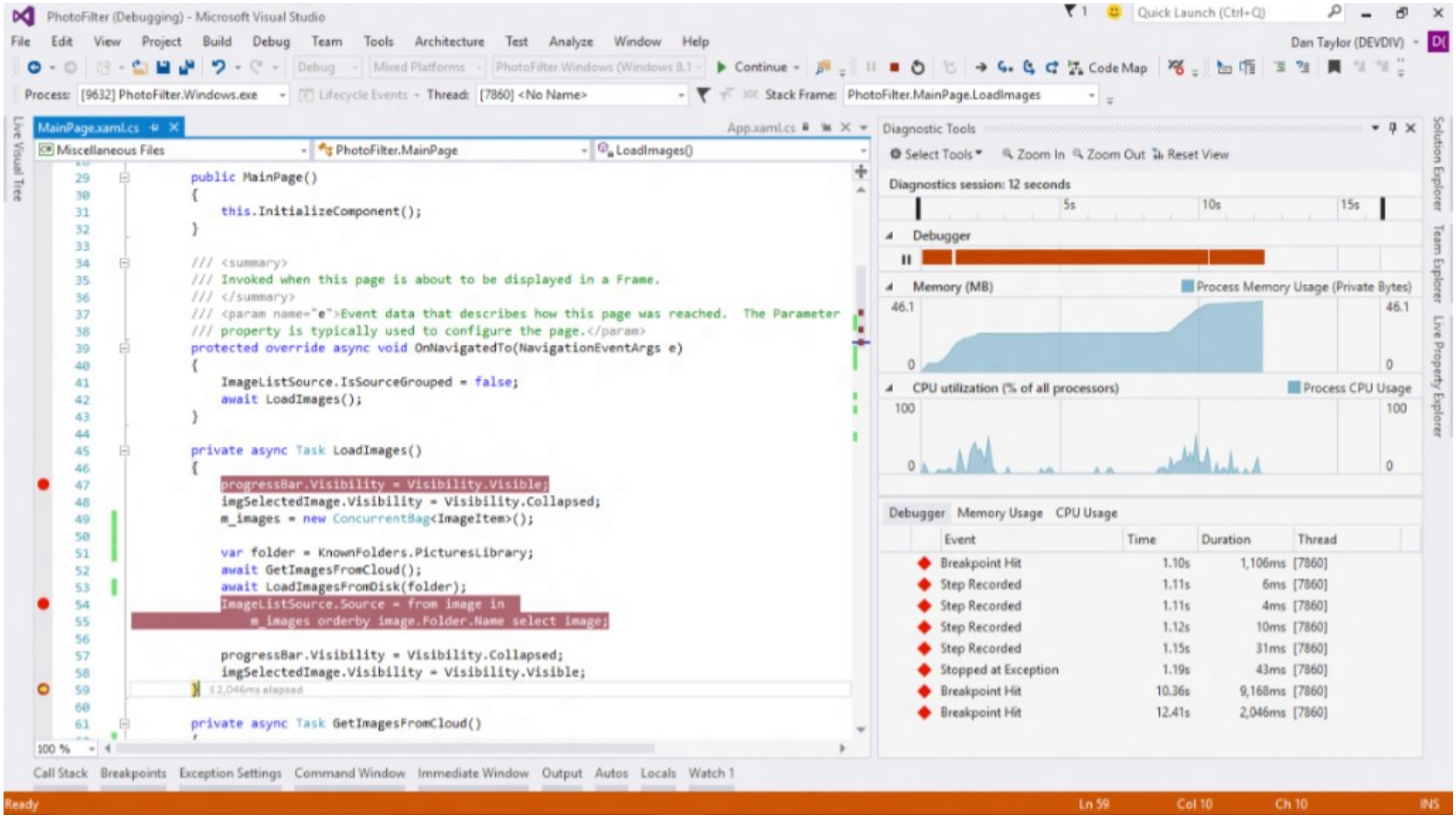
As IDEs facilitam algumas operações comuns no desenvolvimento de aplicações:

- Modelagem - criação do modelo de classes, objetos, interfaces, associações e interações dos artefatos envolvidos no software
- Geração de código - com um escopo mais direcionado a templates de código comumente utilizados para solucionar problemas rotineiros
- Distribuição - auxilia no processo de criação do instalador do software, ou outra forma de distribuição, seja discos ou via internet
- Testes Automatizados - realiza testes no software de forma automatizada, com base em scripts ou programas de testes
- Refatoração - consiste na melhoria constante do código-fonte do software, seja na construção de código mais otimizado, mais limpo e/ou com melhor entendimento

NetBeans


NetBeans 5.5
File Edit View Navigate Source Refactor Build Run CVS Tools Window Help
Projects
Files
Runtime
AnagramGame
Source Packages
com.toy.anagrams.lib
WordLibrary.java
com.toy.anagrams.ui
About.java
Anagrams.java
Test Packages
Libraries
Test Libraries
GUIFormExamples
Source Packages
examples
Antenna.java
ContactEditor.java
Find.java
Libraries
Inspector
Form Antenna
Other Components
[JFrame]
jPanel1 [JPanel]
jLabel1 [JLabel]
jLabel2 [JLabel]
jTextField1 [JTextField]
jTextField2 [JTextField]
jCheckBox1 [JCheckBox]
jPanel2 [JPanel]
jButton3 [JButton]
jButton4 [JButton]
Antenna.java
Anagrams.java
About.java
Find.java
ContactEditor.java
Source
Design
Position/Direction
Direction [°]: 140.000
Height [m]: 110.000
Height is Lower Edge (Not Center)
System
Channels: 2
Watts: 12.000
Adjust
Antenna Type: Kathrein 742151
Electrical Downtilt From [°]: 0.000
To: 10.000
Adjust
Polarization: X +45°
Frequency From [MHz]: 943.000
To: 951.000
Adjust
OK
Cancel
Palette
Swing
JLabel
JButton
JToggleButton
JCheckBox
JRadioButton
ButtonGroup
JComboBox
JList
JTextField
JTextArea
JPanel
JTabbedPane
JScrollBar
JScrollPane
JMenuBar
JPopupMenu
JSlider
JProgressBar
JSplitPane
JFormattedTextField
JPasswordField
JSpinner
JSeparator
JTextPane
JEditorPane
JTree
JTable
JToolBar
jCheckBox1 [JCheckBox] - Properties
Properties
Events
Code
Properties
action null
background [236,233,216]
buttonGroup <none>
componentPopupMenu <none>
font Tahoma 11 Plain
foreground [0,0,0]
mnemonic
selected
text Height is Lower Edge (...
toolTipText null
Other Properties
UIClassID CheckBoxUI
actionCommand Height is Lower Edge (...
alignmentX 0.0

Visual Studio



RAD

O termo RAD (Rapid Application Development) é utilizado tanto para ferramentas, linguagens ou metodologias de desenvolvimento que visam acelerar o desenvolvimento de aplicações. A aplicação do mesmo termo para diferentes situações causa certa confusão. Dificilmente se aplica o termo a uma linguagem apenas sem a associação a uma ferramenta IDE com a utilização de componentes reutilizáveis que implementem parte das funcionalidades do programa. O termo também se aplica ao modelo de processo de desenvolvimento de software interativo e incremental que enfatiza um ciclo de desenvolvimento extremamente curto

Frameworks

Com a evolução do desenvolvimento de software se chegou ao conceito de frameworks, que, apesar das várias definições, pode ser visto como um conjunto de códigos (funções e objetos) que implementam funcionalidades para o desenvolvimento de aplicações.

Um framework pode ser desde o “esqueleto” de uma aplicação, com parte das funcionalidades já implementadas e por si só capaz de gerar uma aplicação básica, podendo até incluir uma IDE RAD, ou ser apenas um conjunto de classes e funções integradas que implementam várias funcionalidades necessárias para o desenvolvimento de aplicações mas sem a implementação de uma aplicação básica. Exemplos: .NET, AWT, Qt, Ruby on Rails.