MC 320
Introdução a Java

Prof. Fernando Vanini
IC – UNICAMP
Klais Soluções Ltda

- A linguagem Java e a máquina virtual
- Objetos e Classes
- Encapsulamento, Herança e Polimorfismo
- Primeiro Exemplo

- Principais características:
  - Orientação ao objetos
  - Exceções
  - '*multi-thread*'
  - Liberação automática de memória
  - Independência de Plataforma

Ao se aprender uma nova linguagem de programação, normalmente se estuda

- sintaxe
- tipos de dados (básicos, compostos, definidos pelo usuário)
- operadores
- comandos de controle
- funções
- entrada e saída
- bibliotecas

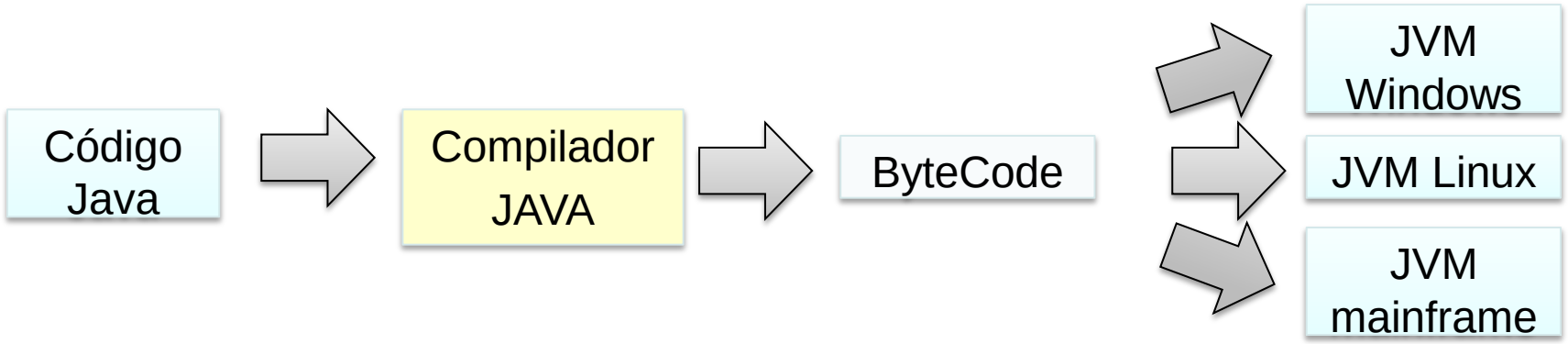
Essas características são comuns à maioria das linguagens ditas "de uso geral". Esse é o caso de C, C++, Pascal, VB e também de C#.

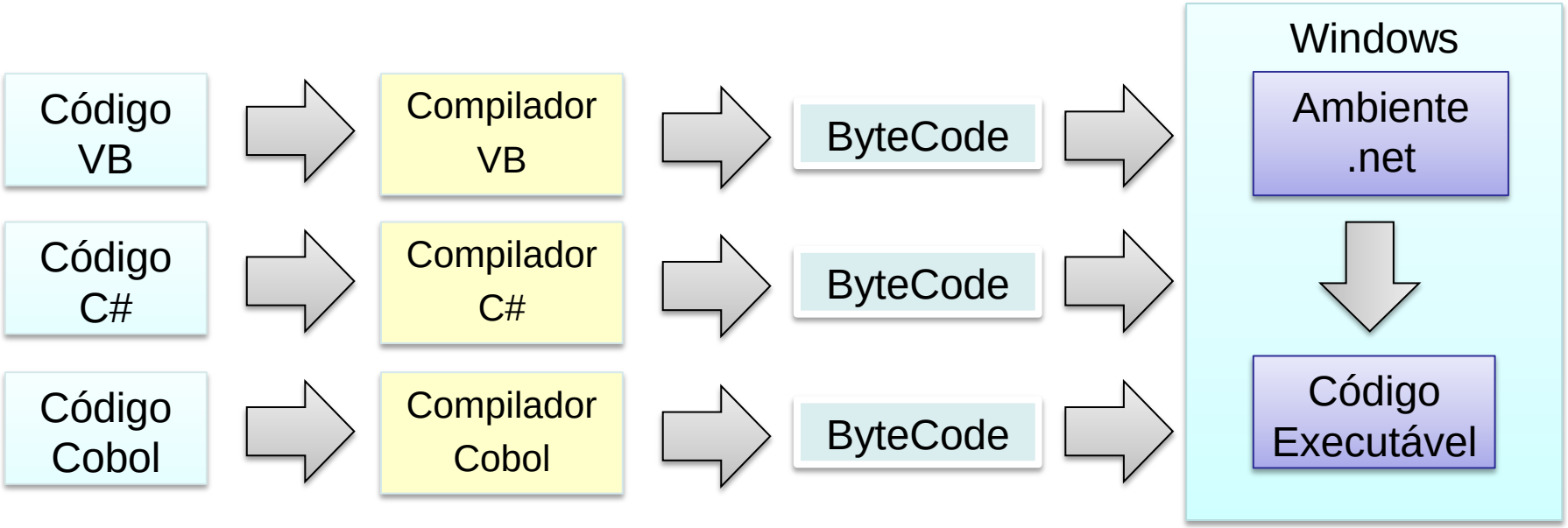
- Um programa, para ser executado, precisa de um conjunto de 'serviços de apoio' que normalmente são oferecidos pelo sistema operacional.
- Em VB, por exemplo, as funções para criação da interface de usuário são exemplos de funções que acessam esses 'serviços de apoio'.
- Normalmente os serviços oferecidos pelo sistema operacional são disponibilizados através de funções de biblioteca.

- O conjunto de serviços oferecidos pelo sistema operacional
  - não é padronizado
  - afeta a estrutura das aplicações
- Essa dependência compromete a portabilidade dos sistemas.
- Por essa razão, alguns fabricantes têm trabalhado na linha de oferecer um 'ambiente de execução' aos programas que seja independente da máquina e do sistema operacional.

- A máquina virtual Java, JVM, constitui o ‘ambiente’ de execução para programas Java.
- É através da JVM que se consegue a independência de plataforma: para cada plataforma (hardware + sistema operacional) é implementada uma versão da JVM.
- Uma vez compilado, o código pode ser executado em qualquer JVM.

JVM x Ambiente .net





Objetos

- Um tipo de dado define
  - um conjunto de valores
  - um conjunto de operações sobre esses valores.
- Uma classe define
  - um conjunto de dados
  - um conjunto de métodos
- Um objeto é um 'elemento' de uma classe
- Cada objeto tem os seus dados associados

Um exemplo

- Suponha uma classe que define números complexos. Cada objeto dessa classe tem associado
  - um valor correspondente à 'parte real'
  - um valor correspondente à 'parte imaginária'
  - um conjunto de operações (soma, subtração, etc.)

Um exemplo

```
public class Complex {
    float r,i;

    public Complex(float rr, float ii){ r = rr; i = ii; }

    public String toString(){ return "("+r+" : "+ i+")"; }

    public Complex add(Complex c) {
      return new Complex(r+c.r, i+c.i);
    }

    public Complex sub(Complex c) {
      return new Complex(r-c.r,i-c.i);
    }

    public Complex mult(Complex c){
      return new Complex( r*c.r-i*c.i, r*c.i+i*c.r );
    }
}
```

Um exemplo

- Nesse exemplo
  - o método

```
public Complex(float rr, float ii){ r = rr; i = ii; }
```

é o construtor dessa classe: utilizado para a criação de objetos dessa classe.

- tem o mesmo nome que a classe
- inicia os atributos do objeto sendo criado (neste caso i e r).
- exemplo de uso: `Complex c = new Complex(1,-1);`

- o modificador **`public`** indica que o método pode ser utilizado por outras classes façam uso dele.

Um exemplo

- Nesse exemplo
  - os métodos add(), sub() e mult() são utilizados para as operações básicas com objetos da classe Complex.

  - Exemplos de uso

```
Complex c = new Complex(1,-1);
Complex d = new Complex(-1,1);
Complex e = d.add(c);  // soma c a d
Complex f = d.mult(c); // d multiplicado por c
```

Um exemplo

- Nesse exemplo
  - as a declaração das 'variáveis'

```
float r,i;
```

define os dados ou *atributos* associados a cada objeto da classe.

  - os atributos definidos dessa forma só podem utilizados pelos métodos da classe onde são definidos.

Um exemplo

- Nesse exemplo
  - o método

```
public string toString(){ return "("+r+" : "+ i+")"; }
```

é responsável pela conversão de um objeto Complex para string. Exemplo:

- Supondo a declaração `Complex c = new Complex(1,-1);`
- ao fazermos `c.toString()`, o valor retornado será o string `"(1 : -1)"`
- o operador + na expressão `"("+r+" : "+ i+")"` está sendo usado para realizar a concatenação de strings.
- o modificador `public` permite que o método `toString()` seja usado em outras classes que utilizam o mesmo espaço de nomes.

Exemplo de uso

```
public class Teste {
  public static void main(String[] args) {
    Complex c1 = new Complex(0.0F, 3.14159F);
    Complex c2 = new Complex(3.14159F,0.0F);
    Complex z  = new Complex(0.0F,0.0F);
    Complex u  = new Complex(1.0F,0.0F);
    Complex c3 = c1.add(c2);
    Complex c4 = c1.sub(c2);
    System.out.println("c1:"+c1.toString()+" c2:"+c2.toString());
    System.out.println("c3:"+c3.toString()+" c4:"+c4.toString());
    System.out.println("c1+z:"+c1.add(z).toString());
    System.out.println("z-c1:"+z.sub(c1).toString());
    System.out.println("z-u:"+z.sub(u).toString());
    System.out.println("c3*u:"+c3.mult(u).toString());
    System.out.println("c1*c2:"+c1.mult(c2).toString());
  }
}
```

Um exemplo

- Nesse exemplo
  - os objetos **c1**, **c2**, **z** e **u** são criados através da chamada ao construtor. Essas variáveis são apenas referências a objetos.
  - quando declaramos uma variável da forma

```
Complex c1;
```

estamos declarando uma referência a um objeto que ainda não está definido.

ao fazermos a atribuição

```
c1 = new Complex(0.00F,3.14159F);
```

estamos indicando que **c1** passa a se referir ao objeto criado através do construtor. Se fizermos uma atribuição da forma

```
c2 = c1;
```

fazemos com que **c1** e **c2** sejam referências aos mesmos objetos.