Programação Orientada a Objetos

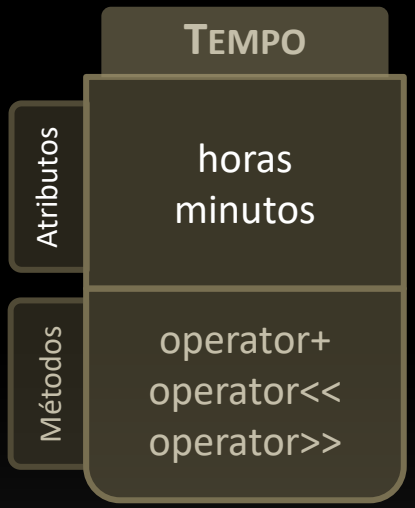
PROJETANDO CLASSES em C++

Introdução

- As nossas **classes** agora podem utilizar:
  - Sobrecarga de operadores
  - Funções amigas

```
class Tempo
{
private:
   int horas;
   int minutos;

public:
   ...

   Tempo operator+(int num) const;
   friend Tempo operator+(int num, const Tempo & t);
};
```

```
Tempo A, B;

// A.operator+(2);
B = A + 2;

// operator+(2, A);
B = 2 + A;
```



Introdução

- As **classes** em C++ são diferentes
  - Se comportam como tipos primitivos
  - Nem toda linguagem OO tem esse suporte

```
Tempo A, B;

// A.operator+(2);
B = A + 2;

// operator+(2, A);
B = 2 + A;
```

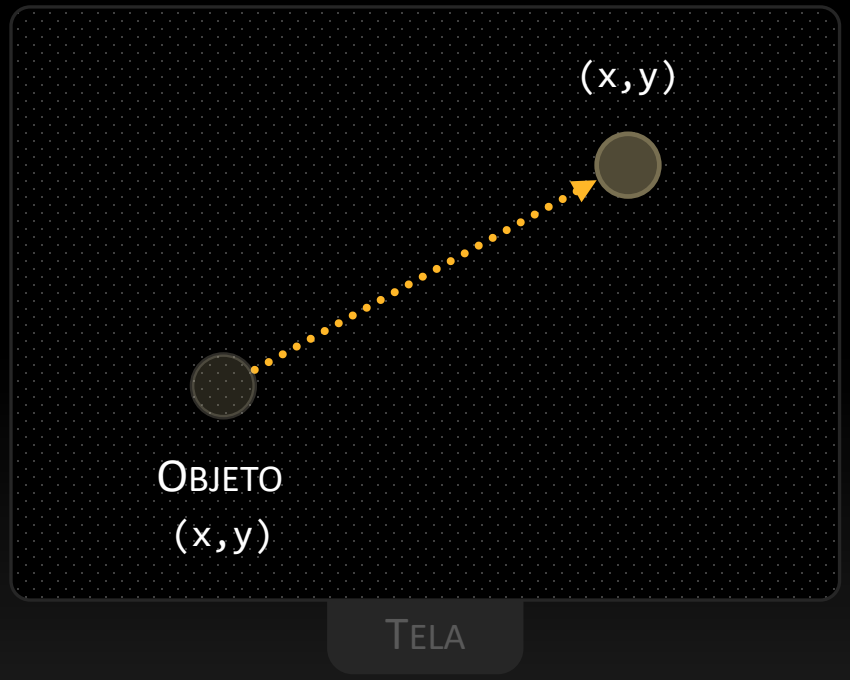
C++

```
Tempo A, B;

// método somar
B = A.Somar(2);

// não tem opção
B = A.Somar(2);
```

Outras

Introdução

- Vamos explorar mais o projeto de classes em C++



Em jogos, vetores são muito usados para deslocar objetos na tela.

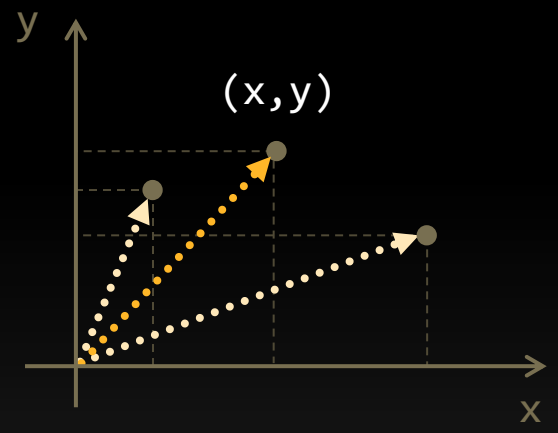
Que tal modelar uma

CLASSE VETOR

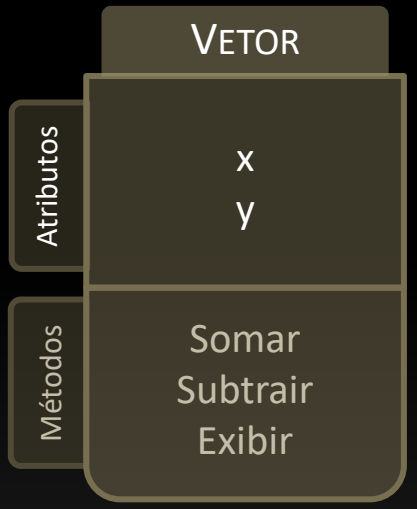
?

Representação de Vetores

- Um vetor pode ser representado por:
  - Coordenadas retangulares
    - Ponto no eixo cartesiano

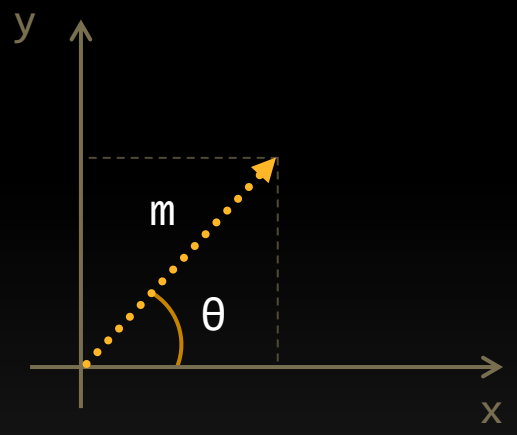



```
class Vetor
{
private:
   double x;
   double y;
   ...
};
```

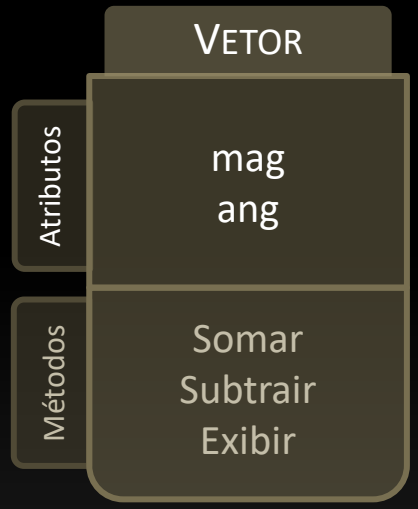


Representação de Vetores

- Um vetor pode ser representado por:
  - Coordenadas polares
    - Magnitude e ângulo

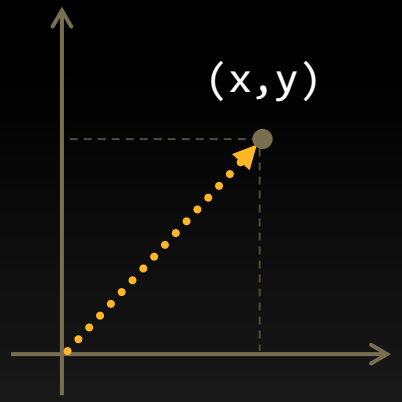


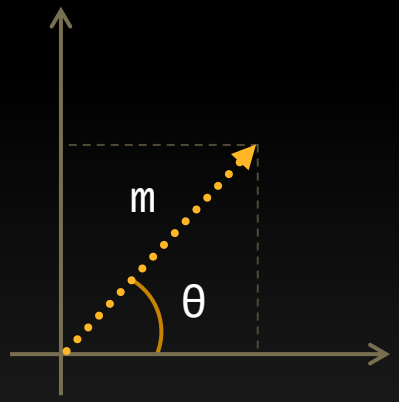

```
class Vetor
{
private:
   double mag;
   double ang;
   ...
};
```



Representação de Vetores

- Podemos guardar as duas representações:
  - Um membro vai indicar a representação padrão
    - Coordenadas Retangulares (RET)
    - Coordenadas Polares (POL)





Representação de Vetores

- A declaração da classe:

```
class Vetor
{
public:
   enum Coord { RET, POL };

private:
   double x, y;      // coordenadas cartesianas
   double mag;       // comprimento do vetor
   double ang;       // ângulo do vetor em graus
   Coord rep;        // representação padrão

   void SetMag();    // ajusta magnitude com base em (x,y)
   void SetAng();    // ajusta ângulo com base em (x,y)
   void SetX();      // ajusta posição x com base em magnitude e ângulo
   void SetY();      // ajusta posição y com base em magnitude e ângulo
   ...
```

Representação de Vetores

```
...

public:
    Vetor();
    Vetor(double n1, double n2, Coord tipo = RET);

    double Magnitude()           { return mag; }
    double Angulo()              { return ang; }
    void   SetCoord(Coord modo)  { rep = modo; }

    Vetor operator+(const Vetor& v) const;
    Vetor operator-(const Vetor& v) const;
    Vetor operator-() const;
    Vetor operator*(double n) const;
    friend Vetor operator*(double n, const Vetor& v);
    friend ostream& operator<<(ostream& os, const Vetor& v);

};
```

Construtores

Métodos Inline

Operações

Representação de Vetores

- Implementação dos construtores

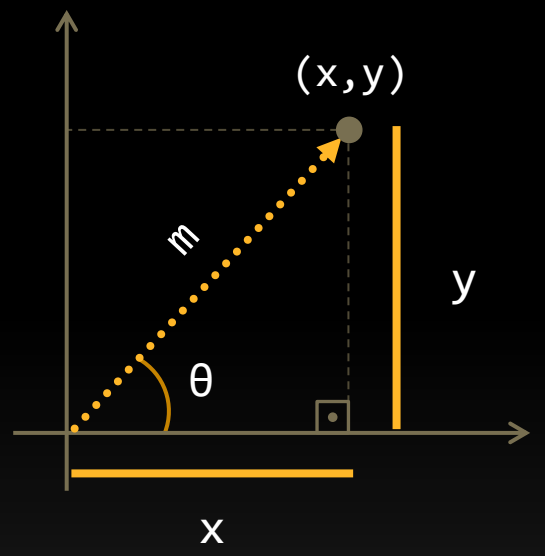
```
// definição da classe Vetor
#include <iostream>
#include <cmath>
#include "Vetor.h"
using std::ostream;

const double GrausPorRad =
    45.0 / atan(1.0);

Vetor::Vetor()
{
    x = 0;
    y = 0;
    ang = 0;
    mag = 0;
    rep = RET;
}
```

```
Vetor::Vetor(double n1, double n2, Coord modo)
{
    rep = modo;
    if (rep == RET)
    {
        x = n1; y = n2;
        SetMag();        // ajusta magnitude
        SetAng();        // ajusta ângulo
    }
    else
    {
        mag = n1;
        ang = n2 / GrausPorRad;
        SetX();
        SetY();
    }
}
```

Representação de Vetores

- Os métodos privados SetMag e SetAng

```
void Vetor::SetMag()
{
    // sqrt = raiz quadrada
    mag = sqrt(x * x + y * y);
}


void Vetor::SetAng()
{
    // atan2 = arco tangente
    ang = atan2(y, x);
}
```



$$m = \sqrt{x^2 + y^2}$$

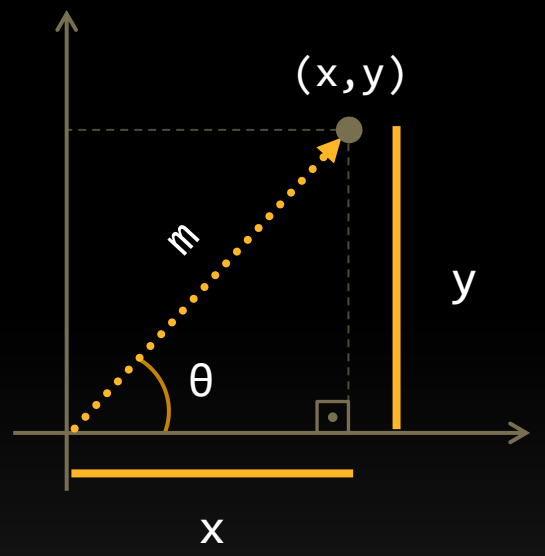
$$\tan\theta = \frac{y}{x}$$

$$\theta = \operatorname{atan}\left(\frac{y}{x}\right)$$

Representação de Vetores

- Os métodos privados `SetX` e `SetY`

```
void Vetor::SetX()
{
    // cos = cosseno
    x = mag * cos(ang);
}


void Vetor::SetY()
{
    // sin = seno
    y = mag * sin(ang);
}
```

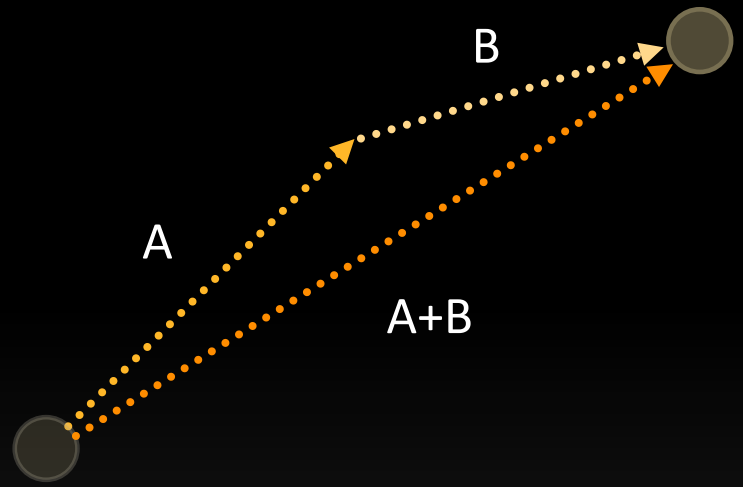


$$\cos\theta = \frac{x}{m}$$

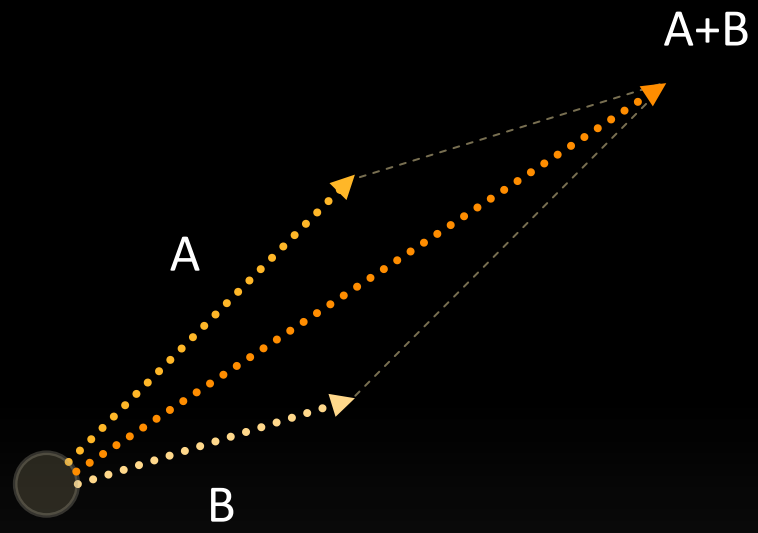
$$\sin\theta = \frac{y}{m}$$

Operações com Vetores

- Soma de vetores é importante para calcular
  - Deslocamentos
  - Forças



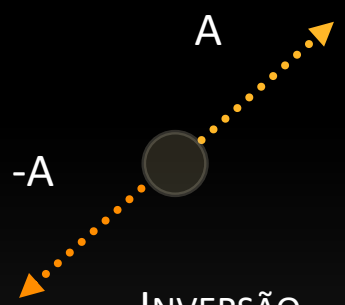
SOMA DE DESLOCAMENTOS



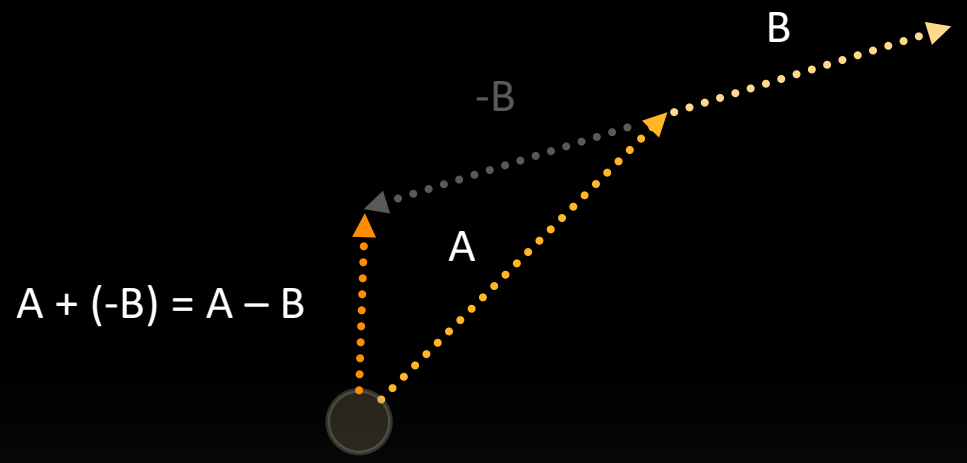
SOMA DE FORÇAS

Operações com Vetores

- A subtração e inversão de vetores
  - Deslocamento com mudança de direção
  - Mudança de direção



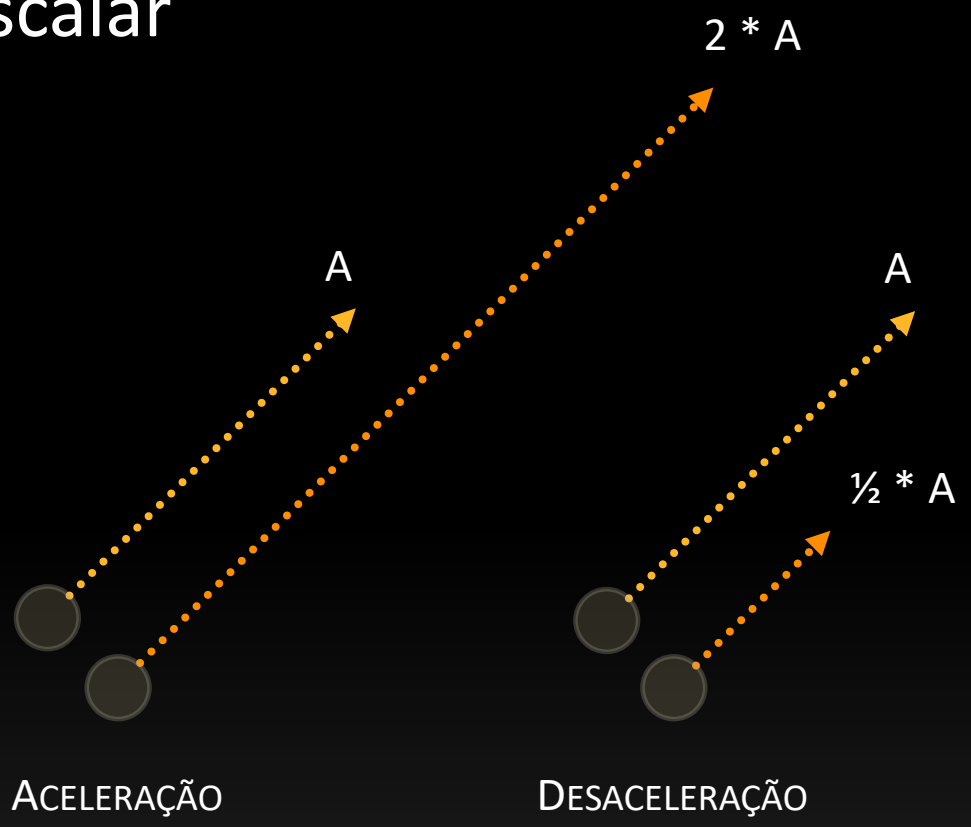
INVERSÃO



SUBTRAÇÃO

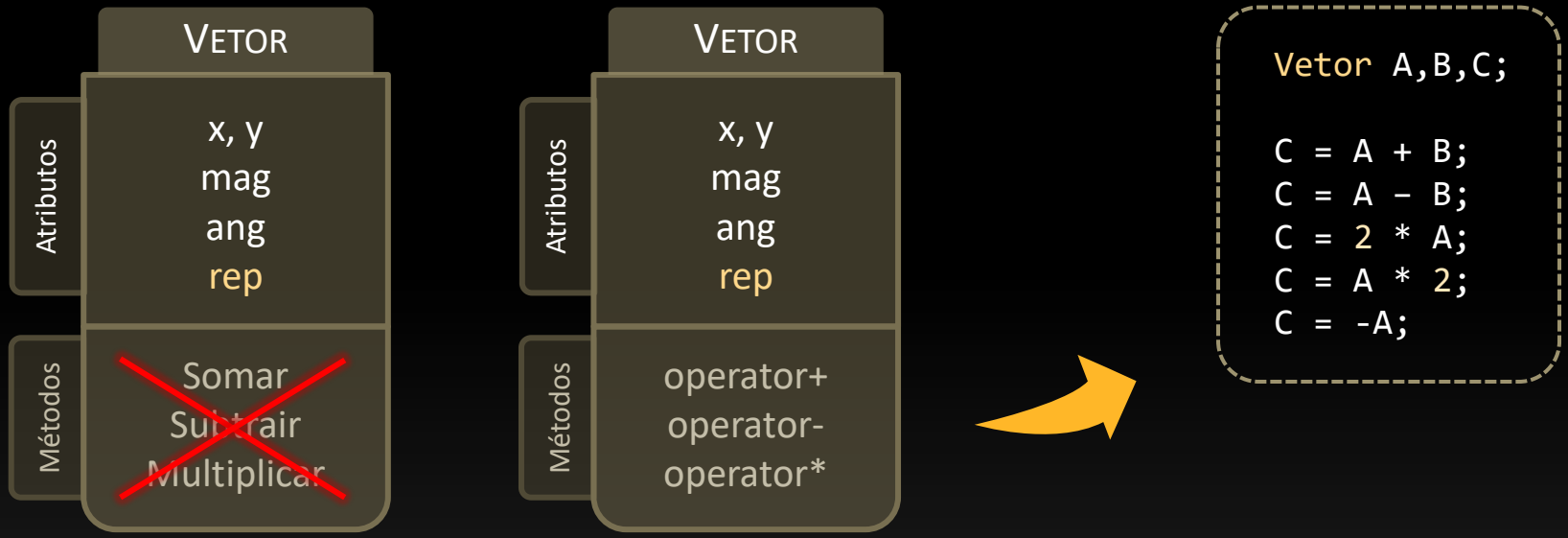
Operações com Vetores

- Multiplicação por escalar
  - Aceleração
  - Desaceleração



Operações com Vetores

- Operações podem ser implementadas por métodos
  - C++ suporta a sobrecarga de operadores



Soma de Vetores

- Implementação:

```
Vetor Vetor::operator+(const Vetor & v) const
{
    Vetor soma;
    soma.x = x + v.x;
    soma.y = y + v.y;
    return soma;

    // não atualizou coordenadas polares
    // poderia fazer isso aqui, mas
    // existe uma forma melhor
}

Vetor Vetor::operator+(const Vetor & v) const
{
    // return { x + v.x, y + v.y };
    return Vetor(x + v.x, y + v.y);
}
```

Se um método precisa criar um novo objeto, verifique se é possível usar o construtor da classe.

```
Vetor A,B,C;

C = A + B;
```

Multiplicação Por Escalar

- Implementação:

```
// função membro
Vetor Vetor::operator*(double n) const
{
    return Vetor(n * x, n * y);
}
```

```
// função amiga
Vetor operator*(double n, const Vetor & v)
{
    return Vetor(n * v.x, n * v.y);
}
```

Requer o uso de função amiga para tratar as duas opções.

```
Vetor A,B,C;

C = A * 2;
C = 2 * A;
```

Subtração e Inversão

- A subtração recebe dois operandos

```
// operando esquerdo é passado implicitamente
Vetor Vetor::operator-(const Vetor & v) const
{
    return Vetor(x - v.x, y - v.y);
}
```

- A inversão recebe apenas um

```
// operando é o objeto passado implicitamente
Vetor Vetor::operator-() const
{
    return Vetor(-x, -y);
}
```

A subtração já é **sobrecarregada** com o menos unário.

```
Vetor A,B,C;

C = A - B;
C = -C;
```

Exibição de Vetores

- Exibe coordenadas retangulares ou polares
  - A função não é membro da classe
    - É preciso usar Vetor::RET e não apenas RET

```
ostream & operator<<(ostream & os, const Vetor & v)
{
    if (v.rep == Vetor::RET)
    {
        os << "(x,y) = (" << v.x << ", " << v.y << ")";
    }
    else
    {
        os << "(m,a) = (" << v.mag << ", " << v.ang * GrausPorRad << ")";
    }
    return os;
}
```

```
class Vetor
{
public:
   enum Coord {RET, POL};
   ...
};
```

Exibição de Vetores

- Altera tipo padrão da coordenada
  - Coordenada retangular
  - Coordenada polar

```
void Vetor::SetCoord(Coord modo)
{  rep = modo; }
```

```
Vetor A { 10, 10 };
cout << A;                  // (x,y) = (10, 10)

A.SetCoord(Vetor::POL);     // (m,a) = (14.1421, 45)
cout << A;
```

Exibição de Vetores

- Observe que o construtor usa Vetor::RET

```
int main()
{
    Vetor A { 10, 10 };
    Vetor B { 20, 20 };
    cout << A;                  // (x,y) = (10, 10)
    cout << B;                  // (x,y) = (20, 20)

    A.SetCoord(Vetor::POL);     // coordenadas polares
    B.SetCoord(Vetor::POL);     // coordenadas polares

    cout << A + B;              // (x,y) = (30, 30)

}
```

```
Vetor Vetor::operator+(const Vetor & v) const
{ return Vetor(x + v.x, y + v.y); }
```

Construtor é usado para criar um novo objeto cuja representação padrão é RET

Exibição de Vetores

- Provavelmente o formato de apresentação não deve ser atrelado aos objetos da classe
  - O que queremos é alternar o formato de exibição

```
cout << Vetor::RET;       // modifica formato
cout << A + B ;           // (x,y) = (30, 30)

cout << Vetor::POL;       // modifica formato
cout << A + B ;           // (m,a) = (42.4264, 45)
```

  - Mas isso ainda não é possível
  - Abordaremos no futuro

Resumo

- O projeto de classes em C++ pode criar **objetos** que:
  - Utilizam a chamada tradicional de métodos
  - Se comportam como tipos primitivos

```
Vetor A,B,C;

C = A.Somar(B);
C = A.Subtrair(B);
C = A.Multiplicar(2);
C = A.Multiplicar(2);
C = A.Inverter();
```

```
Vetor A,B,C;

C = A + B;
C = A – B;
C = 2 * A;
C = A * 2;
C = -A;
```