



Universidade Federal do ABC  
Centro de Matemática, Computação e Cognição

# Programação Orientada a Objetos

Monael Pinheiro Ribeiro, D.Sc.

# Objetos e Métodos Constantes

- **Objetos const**

- Quando um objeto é criado com o qualificador const significa que o objeto é constante.
- Ou seja, nenhum de seus atributos pode ser modificado.
- Por isso, o compilador impede que qualquer método seja acessado sob esse objeto

```
const Data natal(25,12,2053);  
natal.printData("AADDMM", '-');
```

# Objetos e Métodos Constantes

- **Objetos const**

- Quando um objeto é criado com o qualificador const significa que o objeto é constante.
- Ou seja, nenhum de seus atributos pode ser modificado.
- Por isso, o compilador impede que qualquer método seja acessado sob esse objeto

```
const Data natal(25,12,2053);  
natal.printData("AADDMM", '-');
```

**Data.cpp:105:34: error: passing 'const Data' as 'this' argument of 'void Data::printData(std::string, char)' discards qualifiers [-fpermissive]      natal.printData("AADDMM", '-');**

# Objetos e Métodos Constantes

- **Métodos const**

- Nos objetos declarados como const, o compilador impede que qualquer método seja acessado sob esse objeto, pois o compilador não sabe quais métodos o modifica.
- Mas, o programador pode informar quais métodos não modificam o objeto, possibilitando assim que estes métodos operem sobre objetos constantes.

# Objetos e Métodos Constantes

- **Métodos const**

- Exemplo em C++: 

```
class Data
{
```

```
    private:
```

```
        int dia, mes, ano;
```

```
    public:
```

```
        Data();
```

```
        Data(int, int, int);
```

```
        void setDia(int);
```

```
        void setMes(int);
```

```
        void setAno(int);
```

```
        int getDia() const;
```

```
        int getMes() const;
```

```
        int getAno() const;
```

```
        void printData() const;
```

```
        void printData(std::string) const;
```

```
        void printData(std::string, char) const;
```

```
};
```



# Objetos e Métodos Constantes

- **Métodos const**

- Exemplo em C++:

```
int Data::getDia() const
{
    return this->dia;
}
```

```
int Data::getMes() const
{
    return this->mes;
}
```

```
int Data::getAno() const
{
    return this->ano;
}
```

```
void Data::printData() const
{
    this->printData("DDMMAA", '/');
}
```

```
void Data::printData(std::string formato) const
{
    this->printData(formato, '/');
}
```



# Objetos e Métodos Constantes

- **Objetos e Métodos const em JAVA**
  - JAVA não dispõe do modificador const e nem de um equivalente.
  - Mas oferece formas intentar imitar o conceito de Objetos const do C++.

# Objetos e Métodos Constantes

- **Objetos e Métodos const em JAVA**

- JAVA não dispõe do modificador const e nem de um equivalente.
- Mas oferece formas intentar imitar o conceito de Objetos const do C++.





# Atributos e Métodos de Classe

- **Atributos de Classe**

- São atributos com seus respectivos valores presentes em todas as instâncias de uma classe.
- É como um atributo (variável) externo disponível para todos os objetos de uma mesma classe.
- Quando um atributo de classe é criado uma única região de memória é alocada e seu conteúdo é compartilhado por todos os objetos.
- A modificação de um atributo de classe reflete em todos os objetos instanciados daquela classe.
- Para se declarar um atributo de classe se usa o modificador `static` em sua declaração na classe.

# Atributos e Métodos de Classe

- **Atributos de Classe**

- Exemplo em C++:

```
class Ponto
{
    private:
        float x, y;
    public:
        Ponto();
        Ponto(float, float);
        /*metodos sets e gets*/
};
```

```
class Bola
{
    private:
        Ponto posicao;
        float peso, diametro;
    public:
        static float gravidade;

        Bola(Ponto, float, float);
        Bola(float, float);
        /*metodos sets e gets*/
};
```

# Atributos e Métodos de Classe

- **Atributos de Classe**

- Exemplo em C++:

```
float Bola::gravidade = 10;
```

```
Bola::Bola(Ponto pos, float p, float d)
{
    this->setPosicao(pos);
    this->setPeso(p);
    this->setDiametro(d);
}
```

```
Bola::Bola(float p, float d)
{
    this->setPosicao(Ponto(0,0));
    this->setPeso(p);
    this->setDiametro(d);
}
```

# Atributos e Métodos de Classe

- **Atributos de Classe**

- Exemplo em C++:

```
float Bola::gravidade = 10;
```

**Atribuindo valor para o atributo de classe.**

```
Bola::Bola(Ponto pos, float p, float d)
{
    this->setPosicao(pos);
    this->setPeso(p);
    this->setDiametro(d);
}
```


```
Bola::Bola(float p, float d)
{
    this->setPosicao(Ponto(0,0));
    this->setPeso(p);
    this->setDiametro(d);
}
```

# Atributos e Métodos de Classe

- **Atributos de Classe**

- Exemplo em C++:

```
int main()
{
    Bola jabulani(230, 45), brazuca(Ponto(12.32f, 21.43f), 225, 48);
    std::cout << "Jabulani ";
    std::cout << "Gravidade: " << jabulani.getGravidade() << std::endl;
    std::cout << "Brazuca ";
    std::cout << "Gravidade: " << brazuca.getGravidade() << std::endl;

    jabulani.gravidade = 15; 

    std::cout << "Jabulani ";
    std::cout << "Gravidade: " << jabulani.getGravidade() << std::endl;
    std::cout << "Brazuca ";
    std::cout << "Gravidade: " << brazuca.getGravidade() << std::endl;
    return 0;
}
```


```
Jabulani Gravidade: 10
Brazuca Gravidade: 10
Jabulani Gravidade: 15
Brazuca Gravidade: 15
```

# Atributos e Métodos de Classe

- **Atributos de Classe**

- Exemplo em C++:

```
int main()
{
    Bola jabulani(230, 45), brazuca(Ponto(12.32f, 21.43f), 225, 48);
    std::cout << "Jabulani ";
    std::cout << "Gravidade: " << jabulani.getGravidade() << std::endl;
    std::cout << "Brazuca ";
    std::cout << "Gravidade: " << brazuca.getGravidade() << std::endl;

    brazuca.gravidade = 20; 

    std::cout << "Jabulani ";
    std::cout << "Gravidade: " << jabulani.getGravidade() << std::endl;
    std::cout << "Brazuca ";
    std::cout << "Gravidade: " << brazuca.getGravidade() << std::endl;
    return 0;
}
```

```
Jabulani Gravidade: 10
Brazuca Gravidade: 10
Jabulani Gravidade: 20
Brazuca Gravidade: 20
```

# Atributos e Métodos de Classe

- **Atributos de Classe**

- Exemplo em C++:

```
int main()
{
    Bola jabulani(230, 45), brazuca(Ponto(12.32f, 21.43f), 225, 48);
    std::cout << "Jabulani ";
    std::cout << "Gravidade: " << jabulani.getGravidade() << std::endl;
    std::cout << "Brazuca ";
    std::cout << "Gravidade: " << brazuca.getGravidade() << std::endl;

    Bola::gravidade = 12;

    std::cout << "Jabulani ";
    std::cout << "Gravidade: " << jabulani.getGravidade() << std::endl;
    std::cout << "Brazuca ";
    std::cout << "Gravidade: " << brazuca.getGravidade() << std::endl;
    return 0;
}
```



```
Jabulani Gravidade: 10
Brazuca Gravidade: 10
Jabulani Gravidade: 12
Brazuca Gravidade: 12
```

# Atributos e Métodos de Classe

- **Atributos de Classe**

- Exemplo em JAVA:

```
class Ponto
{
    private float x, y;
    public Ponto(float xp, float yp)
    { ... }
    /* metodos sets e gets */
}
```

```
public class Bola
{
    private Ponto posicao;
    private float peso, diametro;

    public static float gravidade = 10;

    public Bola(Ponto pos, float p, float d)
    {
        this.setPosicao(pos);
        this.setPeso(p);
        this.setDiametro(d);
    }

    public Bola(float p, float d)
    {
        this(new Ponto(0,0), p, d);
    }

    /* metodos sets e gets */

    /* metodo main */
}
```




# Atributos e Métodos de Classe

- **Atributos de Classe**

- Exemplo em JAVA:

```
class Ponto
{
    private float x, y;
    public Ponto(float xp, float yp)
    { ... }
    /* metodos sets e gets */
}
```

```
public class Bola
{
    private Ponto posicao;
    private float peso, diametro;

    public static float gravidade = 10; 

    public Bola(Ponto pos, float p, float d)
    {
        this.setPosicao(pos);
        this.setPeso(p);
        this.setDiametro(d);
    }

    public Bola(float p, float d)
    {
        this(new Ponto(0,0), p, d);
    }

    /* metodos sets e gets */

    /* metodo main */
}
```

# Atributos e Métodos de Classe

- **Atributos de Classe**

- Exemplo em JAVA:

```
public static void main(String[] args)
{
    Bola jabulani = new Bola(230, 45);
    Bola brazuca = new Bola(new Ponto(12.32f, 21.43f), 225, 48);
    System.out.print("Jabulani ");
    System.out.println("Gravidade: " + jabulani.getGravidade());
    System.out.print("Brazuca ");
    System.out.println("Gravidade: " + brazuca.getGravidade());

    Bola.gravidade = 12;

    System.out.print("Jabulani ");
    System.out.println("Gravidade: " + jabulani.getGravidade());
    System.out.print("Brazuca ");
    System.out.println("Gravidade: " + brazuca.getGravidade());
}
```

# Atributos e Métodos de Classe

- **Atributos de Classe**

- Exemplo em JAVA:

```
public static void main(String[] args)
{
    Bola jabulani = new Bola(230, 45);
    Bola brazuca = new Bola(new Ponto(12.32f, 21.43f), 225, 48);
    System.out.print("Jabulani ");
    System.out.println("Gravidade: " + jabulani.getGravidade());
    System.out.print("Brazuca ");
    System.out.println("Gravidade: " + brazuca.getGravidade());

    Bola.gravidade = 12;

    System.out.print("Jabulani ");
    System.out.println("Gravidade: " + jabulani.getGravidade());
    System.out.print("Brazuca ");
    System.out.println("Gravidade: " + brazuca.getGravidade());
}
```

```
Jabulani Gravidade: 10
Brazuca Gravidade: 10
Jabulani Gravidade: 12
Brazuca Gravidade: 12
```

# Atributos e Métodos de Classe

- **Métodos de Classe**
  - São métodos que podem ser invocados diretamente da classe.
  - Ou seja, não é necessário instanciar um objeto da classe para invocar o método.
  - Isso é uma novidade ?

# Atributos e Métodos de Classe

- **Métodos de Classe**

- São métodos que podem ser invocados diretamente da classe.
- Ou seja, não é necessário instanciar um objeto da classe para invocar o método.
- Isso é uma novidade ?

```
Math.sqrt(delta);
```

```
Math.pow(b,2);
```

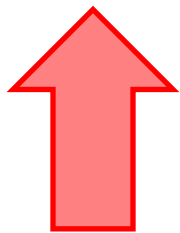
```
System.out.println("Bilu");
```

# Atributos e Métodos de Classe

- **Métodos de Classe**

- São métodos que podem ser invocados diretamente da classe.
- Ou seja, não é necessário instanciar um objeto da classe para invocar o método.
- Isso é uma novidade ?

```
Math.sqrt(delta);  
Math.pow(b,2);  
System.out.println("Bilu");
```



Logo, sqrt() e pow() são métodos de classe de Math  
E println() é método de classe de System.

# Atributos e Métodos de Classe

- **Métodos de Classe**

- São métodos que podem ser invocados diretamente da classe.
- Ou seja, não é necessário instanciar um objeto da classe para invocar o método.
- Isso é uma novidade ?

```
Math.sqrt(delta);
```

```
Math.pow(b,2);
```

```
System.out.println("Bilu");
```

```
var = Scanner.nextInt();
```

# Atributos e Métodos de Classe

- **Métodos de Classe**

- São métodos que podem ser invocados diretamente da classe.
- Ou seja, não é necessário instanciar um objeto da classe para invocar o método.
- Isso é uma novidade ?

```
Math.sqrt(delta);  
Math.pow(b,2);  
System.out.println("Bilu");
```



```
var = Scanner.nextInt();
```



# Atributos e Métodos de Classe

- **Métodos de Classe**

- São métodos que podem ser invocados diretamente da classe.
- Ou seja, não é necessário instanciar um objeto da classe para invocar o método.
- Isso é uma novidade ?

```
Math.sqrt(delta);  
Math.pow(b,2);  
System.out.println("Bilu");
```

```
Scanner scan = new Scanner(System.in);  
var = scan.nextInt();
```



O nextInt() não é um método de classe.  
Logo, necessita-se de um objeto para invoca-lo.

# Atributos e Métodos de Classe

- **Métodos de Classe**

- São métodos que podem ser invocados diretamente da classe.
- Ou seja, não é necessário instanciar um objeto da classe para invocar o método.
- **Os métodos de classe só podem acessar atributos e outros métodos de classe.**
- Para se declarar um método de classe se usa o modificador `static` na declaração de sua assinatura.
- Para invocar um método de classe:
  - Em C++ :
    - `<Nome da Classe>::<Nome do Método>(<arguentos>);`
  - Em JAVA:
    - `<Nome da Classe>.<Nome do Método>(<arguentos>);`

# Atributos e Métodos de Classe

- **Métodos de Classe**

- Exemplo em C++

```
class Ponto
{
    private:
        float x, y;
    public:
        Ponto();
        Ponto(float, float);

        /* metodos sets e gets*/

        static float distancia(Ponto, Ponto);
};
```

# Atributos e Métodos de Classe

- **Métodos de Classe**

- Exemplo em C++

```
float Ponto::distancia(Ponto p, Ponto q)
{
    return std::sqrt(std::pow(q.getX()-p.getX(),2) + std::pow(q.getY()-q.getY(),2));
}
```

# Atributos e Métodos de Classe

- **Métodos de Classe**

- Exemplo em C++

```
float Ponto::distancia(Ponto p, Ponto q)
{
    return std::sqrt(std::pow(q.getX()-p.getX(),2) + std::pow(q.getY()-q.getY(),2));
}

int main()
{
    Ponto p1(2,6), p2(87,10);
    std::cout << "Distancia: " << p1.distancia(p1, Ponto(34,52)) << std::endl;
    std::cout << "Distancia: " << p2.distancia(p1, Ponto(34,52)) << std::endl;
    std::cout << "Distancia: " << Ponto::distancia(p1, Ponto(34,52)) << std::endl;
    return 0;
}
```

# Atributos e Métodos de Classe

- **Métodos de Classe**

- Exemplo em C++

```
float Ponto::distancia(Ponto p, Ponto q)
{
    return std::sqrt(std::pow(q.getX()-p.getX(),2) + std::pow(q.getY()-q.getY(),2));
}

int main()
{
    Ponto p1(2,6), p2(87,10);
    std::cout << "Distancia: " << p1.distancia(p1, Ponto(34,52)) << std::endl;
    std::cout << "Distancia: " << p2.distancia(p1, Ponto(34,52)) << std::endl;
    std::cout << "Distancia: " << Ponto::distancia(p1, Ponto(34,52)) << std::endl;
    return 0;
}
```

```
Distancia: 56.0357
Distancia: 56.0357
Distancia: 56.0357
```

# Atributos e Métodos de Classe

- **Métodos de Classe**

- Exemplo em C++

```
float Ponto::distancia(Ponto p, Ponto q)
{
    return std::sqrt(std::pow(q.getX()-p.getX(),2) + std::pow(q.getY()-q.getY(),2));
}

int main()
{
    Ponto p1(2,6), p2(87,10);
    std::cout << "Distancia: " << p1.distancia(p1, Ponto(34,52)) << std::endl;
    std::cout << "Distancia: " << p2.distancia(p1, Ponto(34,52)) << std::endl;
    std::cout << "Distancia: " << Ponto::distancia(p1, Ponto(34,52)) << std::endl;
    return 0;
}
```

```
Distancia: 56.0357
Distancia: 56.0357
Distancia: 56.0357
```

# Atributos e Métodos de Classe

- **Métodos de Classe**

- Exemplo em JAVA

```
public class Ponto
{
    private float x, y;

    public Ponto(){    }
    public Ponto(float px, float py){ ... }

    /* metodos sets e gets */

    public static double distancia(Ponto p, Ponto q)
    {
        return Math.sqrt(Math.pow(q.getX()-p.getX(),2) + Math.pow(q.getY()-p.getY(),2));
    }

    public static void main(String[] args)
    {
        Ponto p1 = new Ponto(2,6), p2 = new Ponto(1,1);
        System.out.println("Distancia: " + p1.distancia(p1, new Ponto(34,52)));
        System.out.println("Distancia: " + p2.distancia(p1, new Ponto(34,52)));
        System.out.println("Distancia: " + Ponto.distancia(p1, new Ponto(34,52)));
    }
}
```



# Atributos e Métodos de Classe

- **Métodos de Classe**

- Exemplo em JAVA

```
public class Ponto
{
    private float x, y;

    public Ponto(){ }
    public Ponto(float px, float py){ ... }

    /* metodos sets e gets */

    public static double distancia(Ponto p, Ponto q)
    {
        return Math.sqrt(Math.pow(q.getX()-p.getX(),2) + Math.pow(q.getY()-p.getY(),2));
    }

    public static void main(String[] args)
    {
        Ponto p1 = new Ponto(2,6), p2 = new Ponto(1,1);
        System.out.println("Distancia: " + p1.distancia(p1, new Ponto(34,52)));
        System.out.println("Distancia: " + p2.distancia(p1, new Ponto(34,52)));
        System.out.println("Distancia: " + Ponto.distancia(p1, new Ponto(34,52)));
    }
}
```

```
Distancia: 56.0357029044876
Distancia: 56.0357029044876
Distancia: 56.0357029044876
```

# Atributos e Métodos de Classe

- **Métodos de Classe**

- Exemplo em JAVA

```
public class Ponto
{
    private float x, y;

    public Ponto(){    }
    public Ponto(float px, float py){ ... }

    /* metodos sets e gets */

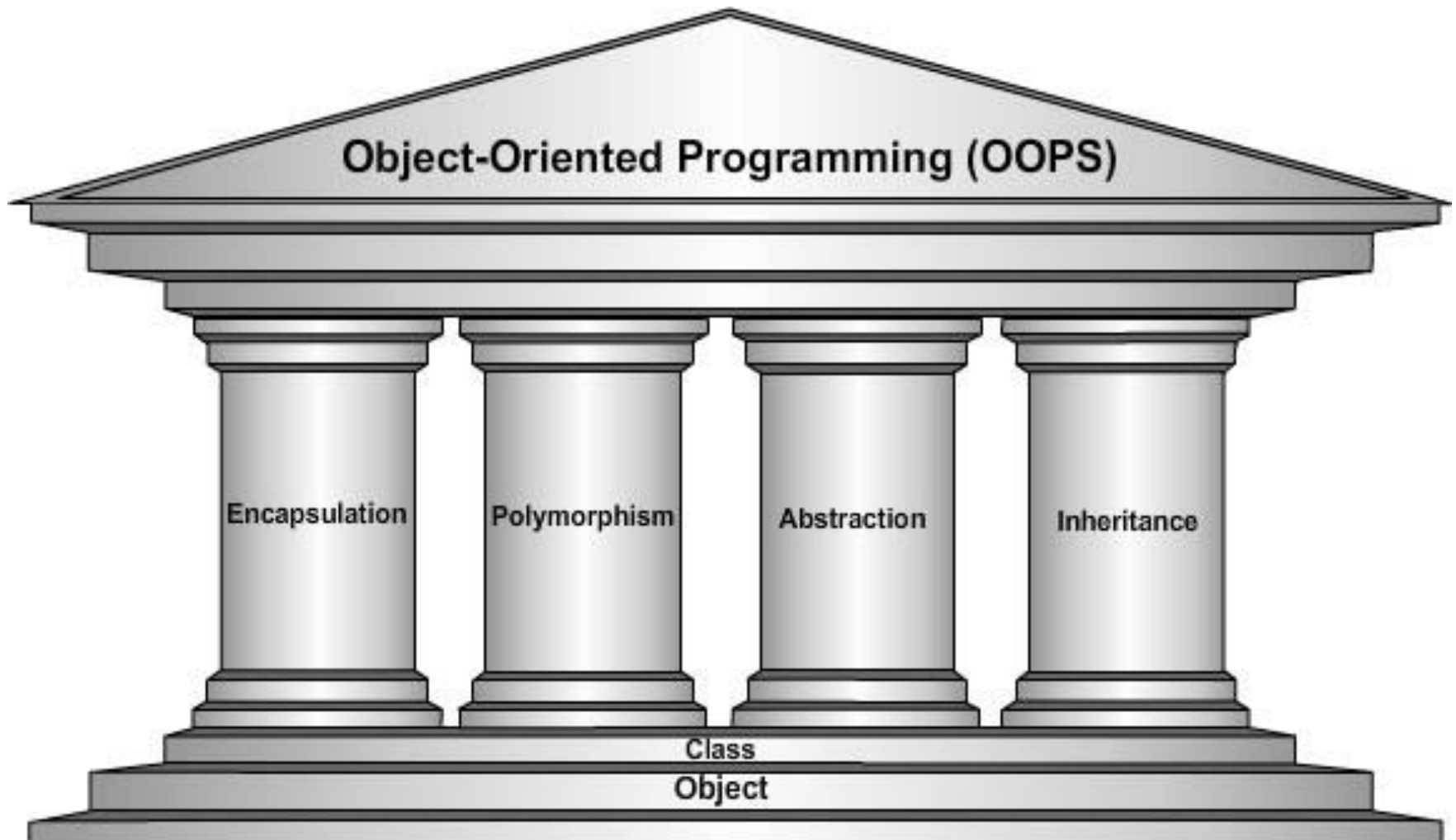
    public static double distancia(Ponto p, Ponto q)
    {
        return Math.sqrt(Math.pow(q.getX()-p.getX(),2) + Math.pow(q.getY()-p.getY(),2));
    }

    public static void main(String[] args)
    {
        Ponto p1 = new Ponto(2,6), p2 = new Ponto(1,1);
        System.out.println("Distancia: " + p1.distancia(p1, new Ponto(34,52)));
        System.out.println("Distancia: " + p2.distancia(p1, new Ponto(34,52)));
        System.out.println("Distancia: " + Ponto.distancia(p1, new Ponto(34,52)));
    }
}
```

```
Distancia: 56.0357029044876
Distancia: 56.0357029044876
Distancia: 56.0357029044876
```

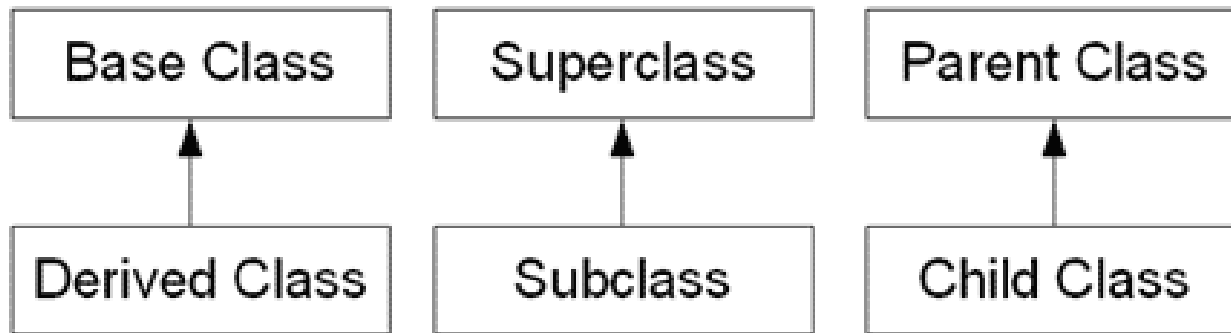


# Os 4 Pilares da POO



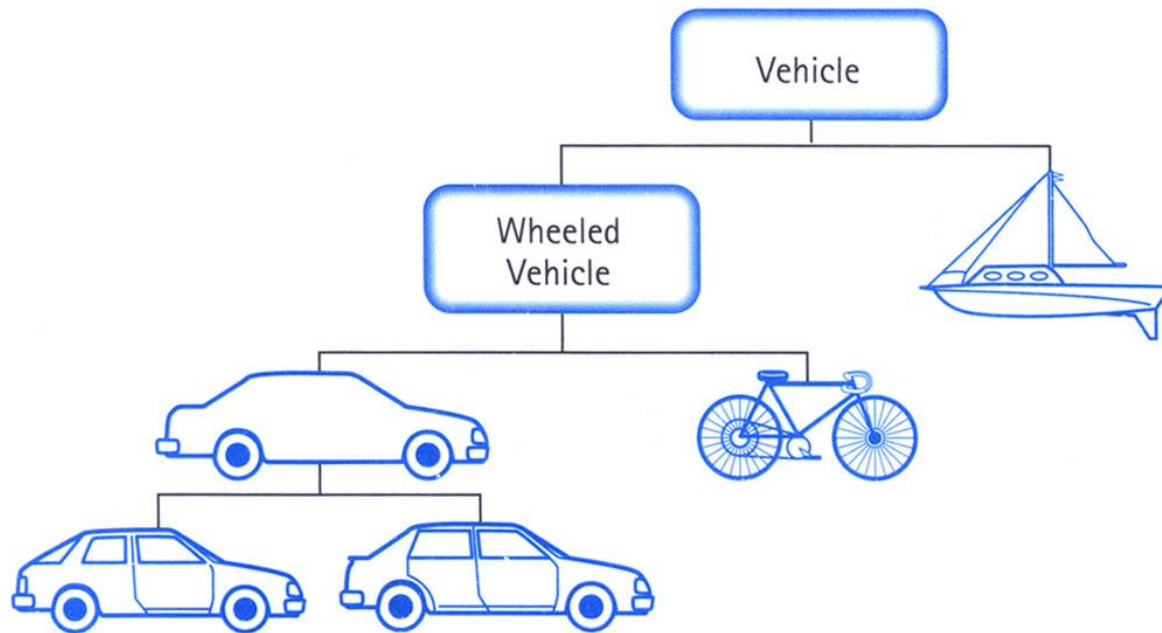
# Herança

- A herança é uma forma de reutilização de código.
- Através da herança cria-se novas classes a partir das classes já existentes.
- Às classes criadas pelo processo de herança dá-se o nome de Classes Derivadas.
- E às classes já existentes dá-se o nome de Classes-Base.



# Herança

- As Classes Derivadas herdam todas as características da Classe-Base.
- E pode acrescentar novas características próprias.



- A maior vantagem da herança é a reutilização de código.
- Facilitando a manutenibilidade, confiabilidade e escalabilidade.
- Uma vez que pode-se criar toda uma hierarquia de classes, sem modificar as classes-base e acrescentar as características próprias das classes derivadas.

# Herança

- Derivando Classes em C++ (Classe Base)

```
class Conta
{
    private:
        std::string numero;
        std::string titular;
        std::string cpf;
        bool bloqueada;
        float saldo;
    public:
        Conta();
        Conta(std::string, std::string);
        void setNumero(std::string);
        void setTitular(std::string);
        void setCpf(std::string);
        void setBloqueada(bool);
        void setSaldo(float);
        std::string getNumero();
        std::string getTitular();
        std::string getCpf();
        bool isBloqueada();
        float getSaldo();
        bool saque(float);
        bool deposito(float);
        void extrato();
        static std::string geraNumero(int);
};
```

```
class Data
{
    private:
        int dia, mes, ano;
    public:
        Data();
        Data(int, int, int);
        void setDia(int);
        void setMes(int);
        void setAno(int);
        int getDia();
        int getMes();
        int getAno();
};

Conta::Conta(std::string cpf, std::string nome)
{
    this->setNumero(Conta::geraNumero(10));
    this->setCpf(cpf);
    this->setTitular(nome);
    this->setSaldo(0);
}
```

# Herança

- Derivando Classes em C++ (Classe Base)

```
bool Conta::saque(float valor)
{
    if(this->getSaldo() >= valor && !this->isBloqueada())
    {
        this->setSaldo(this->getSaldo()-valor);
        return true;
    }
    return false;
}

bool Conta::deposito(float valor)
{
    if(!this->isBloqueada())
    {
        this->setSaldo(this->getSaldo()+valor);
        return true;
    }
    return false;
}

void Conta::extrato()
{
    std::cout << "===== " << std::endl;
    std::cout << "CONTA : " << this->getNumero() << std::endl;
    std::cout << "CPF ..: " << this->getCpf() << std::endl;
    std::cout << "NOME ..: " << this->getTitular() << std::endl;
    std::cout << "SALDO : R$" << this->getSaldo() << std::endl;
    std::cout << "===== " << std::endl;
}
```

# Herança

- Derivando Classes em C++ (Classe Base)

```
int main()
{
    Conta cta("22388645633", "Monael");
    cta.extrato();
    cta.deposito(100);
    cta.extrato();
    return 0;
}
```

```
=====
CONTA : 7935059797
CPF  .: 22388645633
NOME .: Monael
SALDO : R$0
=====
```



# Herança

- Derivando Classes em C++ (Classe Base)

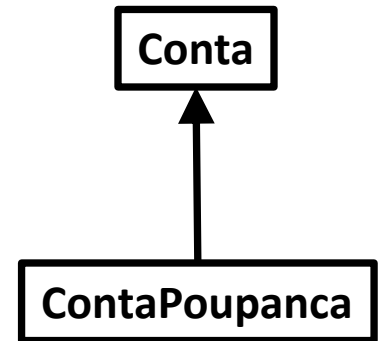
```
int main()
{
    Conta cta("22388645633", "Monael");
    cta.extrato();
    cta.deposito(100);
    cta.extrato();
    return 0;
}
```

```
=====
CONTA : 7935059797
CPF ..: 22388645633
NOME .: Monael
SALDO : R$0
=====
=====
CONTA : 7935059797
CPF ..: 22388645633
NOME .: Monael
SALDO : R$100
=====
```

# Herança

- Derivando Classes em C++ (Classe Derivada)

```
class ContaPoupanca:Conta
{
    private:
        Data aniversario;
        float txRef;
    public:
        ContaPoupanca(std::string, std::string, int, int, int);
        void setAniversario(int, int, int);
        void setAniversario(Data);
        void setTxRef(float);
        Data getAniversario();
        float getTxRef();
        void correcao();
        void extrato();
};
```



# Herança

- Derivando Classes em C++

```
class ContaPoupanca:Conta ← Herança
{
    private:
        Data aniversario;
        float txRef;
    public:
        ContaPoupanca(std::string, std::string, int, int, int);
        void setAniversario(int, int, int);
        void setAniversario(Data);
        void setTxRef(float);
        Data getAniversario();
        float getTxRef();
        void correcao();
        void extrato();
};
```

# Herança

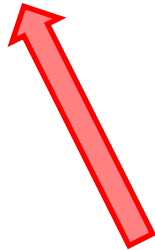
- Derivando Classes em C++

```
class classe_filho: [public | protected | private] classe_pai
```

# Herança

- Derivando Classes em C++

```
class classe_filho: [public | protected | private] classe_pai
```



Herança pública: Indica que os atributos e métodos públicos da classe-pai serão públicos na classe filho. Enquanto os protected da classe-pai continuam como protected na classe filho.

# Herança

- Derivando Classes em C++

```
class classe_filho: [public | protected | private] classe_pai
```



Herança protegida: Indica que os atributos e métodos públicos e protegidos da classe-pai serão protegidos na classe filho.

# Herança

- Derivando Classes em C++

```
class classe_filho: [public | protected | private] classe_pai
```

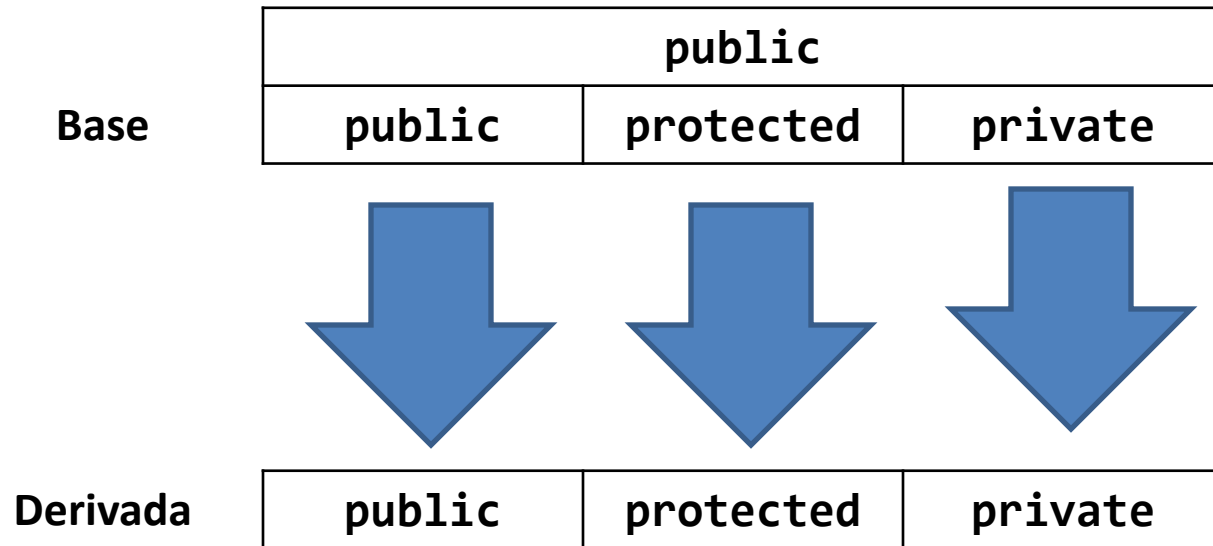


**Herança privada:** Indica que os atributos e métodos públicos e protected da classe-pai serão privados na classe filho. Assim, apenas objetos da classe-pai terão acesso à tais membros, enquanto objetos da classe filho não terão este acesso.

# Herança

- Derivando Classes em C++

```
class classe_filho: [public | protected | private] classe_pai
```

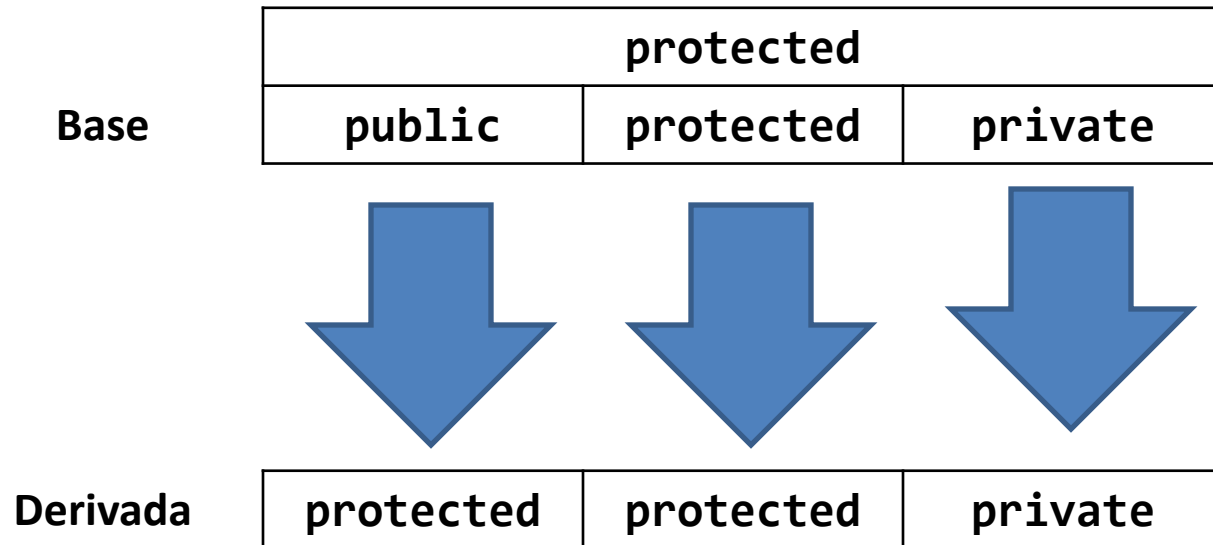




# Herança

- Derivando Classes em C++

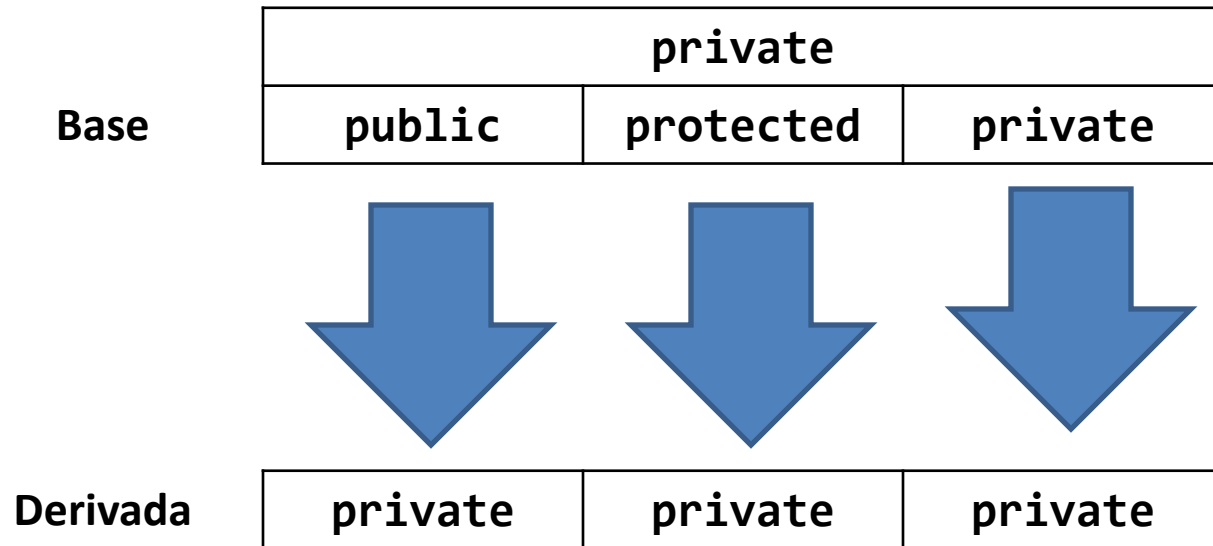
```
class classe_filho: [public | protected | private] classe_pai
```



# Herança

- Derivando Classes em C++

```
class classe_filho: [public | protected | private] classe_pai
```



# Herança

- Derivando Classes em C++

```
class ContaPoupanca:Conta
{
    private:
        Data aniversario;
        float txRef;
    public:
        ContaPoupanca(std::string, std::string, int, int, int);
        void setAniversario(int, int, int);
        void setAniversario(Data);
        void setTxRef(float);
        Data getAniversario();
        float getTxRef();
        void correcao();
        void extrato();
};
```


 **Herança**

Omitir o tipo de herança, implica  
em protected por padrão

# Herança

- Derivando Classes em C++

```
class ContaPoupanca:Conta
{
    private:
        Data aniversario;
        float txRef;
    public:
        ContaPoupanca(std::string, std::string, int, int, int);
        void setAniversario(int, int, int);
        void setAniversario(Data);
        void setTxRef(float);
        Data getAniversario();
        float getTxRef();
        void correcao();
        void extrato();
};
```



Atributos específicos da Subclasse

# Herança

- Derivando Classes em C++

```
class ContaPoupanca:Conta
{
```

```
    private:
```

```
        Data aniversario;
        float txRef;
```

```
    public:
```

```
        ContaPoupanca(std::string, std::string, int, int, int);
        void setAniversario(int, int, int);
        void setAniversario(Data);
        void setTxRef(float);
        Data getAniversario();
        float getTxRef();
        void correcao();
        void extrato();
```

```
};
```

Atributos [private]  
herdados

std::string numero;  
std::string titular;  
std::string cpf;  
bool bloqueada;  
float saldo;

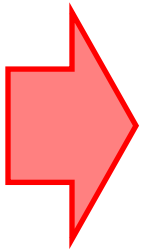
Atributos específicos da Subclasse

# Herança

- Derivando Classes em C++

```
class ContaPoupanca:Conta
{
    private:
        Data aniversario;
        float txRef;
    public:
        ContaPoupanca(std::string, std::string, int, int, int);
        void setAniversario(int, int, int);
        void setAniversario(Data);
        void setTxRef(float);
        Data getAniversario();
        float getTxRef();
        void correcao();
        void extrato();
};
```

Metodos  
específicos  
da  
Subclasse

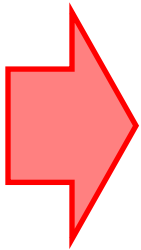


# Herança

- Derivando Classes em C++

```
class ContaPoupanca:Conta
{
    private:
        Data aniversario;
        float txRef;
    public:
        ContaPoupanca(std::string, std::string, int, int, int);
        void setAniversario(int, int, int);
        void setAniversario(Data);
        void setTxRef(float);
        Data getAniversario();
        float getTxRef();
        void correcao();
        void extrato();
};
```

Metodos  
específicos  
da  
Subclasse



Métodos [public]  
herdados

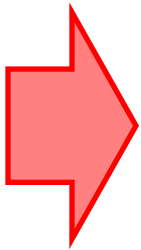
```
Conta();
Conta(std::string, std::string);
void setNumero(std::string);
void setTitular(std::string);
void setCpf(std::string);
void setBloqueada(bool);
void setSaldo(float);
std::string getNumero();
std::string getTitular();
std::string getCpf();
bool isBloqueada();
float getSaldo();
bool saque(float);
bool deposito(float);
void extrato();
static std::string geraNumero(int);
```

# Herança

- Derivando Classes em C++

```
class ContaPoupanca:Conta
{
    private:
        Data aniversario;
        float txRef;
    public:
        ContaPoupanca(std::string, std::string, int, int, int);
        void setAniversario(int, int, int);
        void setAniversario(Data);
        void setTxRef(float);
        Data getAniversario();
        float getTxRef();
        void correcao();
        void extrato();
};
```

Metodos  
específicos  
da  
Subclasse



Métodos [public]  
herdados

```
Conta();
Conta(std::string, std::string);
void setNumero(std::string);
void setTitular(std::string);
void setCpf(std::string);
void setBloqueada(bool);
void setSaldo(float);
std::string getNumero();
std::string getTitular();
std::string getCpf();
bool isBloqueada();
float getSaldo();
bool saque(float);
bool deposito(float);
void extrato();
static std::string geraNumero(int);
```

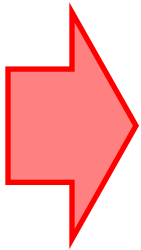


# Herança

- Derivando Classes em C++

```
class ContaPoupanca:Conta
{
    private:
        Data aniversario;
        float txRef;
    public:
        ContaPoupanca(std::string, std::string, int, int, int);
        void setAniversario(int, int, int);
        void setAniversario(Data);
        void setTxRef(float);
        Data getAniversario();
        float getTxRef();
        void correcao();
        void extrato();
};
```

Metodos  
específicos  
da  
Subclasse



Métodos [public]  
herdados

```
Conta();
Conta(std::string, std::string);
void setNumero(std::string);
void setTitular(std::string);
void setCpf(std::string);
void setBloqueada(bool);
void setSaldo(float);
std::string getNumero();
std::string getTitular();
std::string getCpf();
bool isBloqueada();
float getSaldo();
bool saque(float);
bool deposito(float);
void extrato();
static std::string geraNumero(int);
```

## Redefinição de Método

# Herança

- Redefinição de Método
  - Quando herdam-se métodos de uma superclasse, nem sempre esses métodos atendem completamente a necessidade.
  - Pode-se desejar acrescentar algo ao método herdado, ou modificá-lo completamente.
  - Neste caso, pode-se redefinir o método da superclasse na subclasse.
  - Sempre que um objeto da subclasse invocar este método uma busca por ele na subclasse é realizado, se o encontrar ele será executado, caso contrário a busca ascende na hierarquia das classes.
  - Caso deseje invocar o método da superclasse na redefinição da subclasse é possível através do operador de resolução de escopo (::).

# Herança

- Derivando Classes em C++

```
class ContaPoupanca:Conta
{
    private:
        Data aniversario;
        float txRef;
    public:
        ContaPoupanca(std::string, std::string, int, int, int);
        void setAniversario(int, int, int);
        void setAniversario(Data);
        void setTxRef(float);
        Data getAniversario();
        float getTxRef();
        void correcao();
        void extrato();
};
```

# Herança

- Derivando Classes em C++

```
class ContaPoupanca:Conta
{
    private:
        Data aniversario;
        float txRef;
    public:
        ContaPoupanca(std::string, std::string, int, int, int);
        void setAniversario(int, int, int);
        void setAniversario(Data);
        void setTxRef(float);
        Data getAniversario();
        float getTxRef();
        void correcao();
        void extrato();
};

void ContaPoupanca::extrato()
{
    Conta::extrato();
    std::cout << "ANIVERSARIO: " << this->getAniversario().getDia() << std::endl;
    std::cout << "=====" << std::endl;
}
```


# Herança

- Derivando Classes em C++

```
class ContaPoupanca:Conta
{
    private:
        Data aniversario;
        float txRef;
    public:
        ContaPoupanca(std::string, std::string, int, int, int);
        void setAniversario(int, int, int);
        void setAniversario(Data);
        void setTxRef(float);
        Data getAniversario();
        float getTxRef();
        void correcao();
        void extrato();
};

void ContaPoupanca::extrato()
{
    Conta::extrato();
    std::cout << "ANIVERSARIO: " << this->getAniversario().getDia() << std::endl;
    std::cout << "===== " << std::endl;
}
```

**Invocando o método extrato() da classe base.**



# Herança

- Construtores da Classe Derivada
  - Quando a subclasse não define um construtor, o construtor **vazio** da superclasse é invocado.
  - Quando a subclasse define um construtor, o construtor **vazio** da superclasse é invocado e **logo em seguida o seu próprio construtor**.
  - Caso deseje que outro construtor seja invocado, não o construtor vazio, então deve-se explicitar sua invocação.
  - Para invocar um construtor da superclasse a partir do construtor da subclasse acrescente o seguinte comando na definição do corpo do construtor da subclasse:

```
ContaPoupanca::ContaPoupanca(std::string cpf, std::string n, int d, int m, int a):Conta(cpf, n)
{
    this->setAniversario(Data(d, m, a));
}
```

- Neste construtor da subclasse ContaPoupanca, o construtor Conta(std::string, std::string) da superclasse Conta é invocado, e além disso, o atributo específico aniversario da subclasse é inicializado.

# Herança

- Construtores da Classe Derivada
  - Quando a subclasse não define um construtor, o construtor **vazio** da superclasse é invocado.
  - Quando a subclasse define um construtor, o construtor **vazio** da superclasse é invocado e **logo em seguida o seu próprio construtor**.
  - Caso deseje que outro construtor seja invocado, não o construtor vazio, então deve-se explicitar sua invocação.
  - Para invocar um construtor da superclasse a partir do construtor da subclasse acrescente o seguinte comando na definição do corpo do construtor da subclasse:

```
ContaPoupanca::ContaPoupanca(std::string cpf, std::string n, int d, int m, int a):Conta(cpf, n)
{
    this->setAniversario(Data(d, m, a));
}
```

**Invocando o construtor da classe base.**



- Neste construtor da subclasse ContaPoupanca, o construtor Conta(std::string, std::string) da superclasse Conta é invocado, e além disso, o atributo específico aniversario da subclasse é inicializado.

# Herança

- Derivando Classes em C++

```
int main()
{
    ContaPoupanca cta("22388645633", "Monael", 05, 10, 2016);
    cta.extrato();
    cta.deposito(100);
    cta.extrato();
    return 0;
}
```



# Herança

- Derivando Classes em C++

```
int main()
{
    ContaPoupanca cta("22388645633", "Monael", 05, 10, 2016);
    cta.extrato();
    cta.deposito(100);
    cta.extrato();
    return 0;
}
```

```
=====
CONTA : 8620019005
CPF ..: 22388645633
NOME ..: Monael
SALDO : R$0
=====
ANIVERSARIO: dia 5
=====
```

# Herança

- Derivando Classes em C++

```
int main()
{
    ContaPoupanca cta("22388645633", "Monael", 05, 10, 2016);
    cta.extrato();
    cta.deposito(100);
    cta.extrato();
    return 0;
}
```

**Invocando o método depósito(float)  
que fora definido na classe base.**

```
=====
CONTA : 8620019005
CPF ..: 22388645633
NOME ..: Monael
SALDO : R$0
=====
ANIVERSARIO: dia 5
=====
```

# Herança

- Derivando Classes em C++

```
int main()
{
    ContaPoupanca cta("22388645633", "Monael", 05, 10, 2016);
    cta.extrato();
    cta.deposito(100);
    cta.extrato();
    return 0;
}
```

```
=====
CONTA : 8620019005
CPF ..: 22388645633
NOME ..: Monael
SALDO : R$100
=====
ANIVERSARIO: dia 5
=====
```

# Herança

- Derivando Classes em JAVA

```
class ContaPoupanca extends Conta
{
    private Data aniversario;
    private float txRef;


    public ContaPoupanca(String cpf, String nome, int d, int m, int a)
    { ... }
    public void setAniversario(int d, int m, int a)
    { ... }
    public void setAniversario(Data dt)
    { ... }
    public void setTxRef(float tx)
    { ... }
    public Data getAniversario()
    { ... }
    public float getTxRef()
    { ... }
    public void correcao()
    { ... }
    public void extrato()
    { ... }
}
```

# Herança

- Derivando Classes em JAVA

```
class ContaPoupanca extends Conta
{
    private Data aniversario;
    private float txRef;

    public ContaPoupanca(String cpf, String nome, int d, int m, int a)
    { ... }
    public void setAniversario(int d, int m, int a)
    { ... }
    public void setAniversario(Data dt)
    { ... }
    public void setTxRef(float tx)
    { ... }
    public Data getAniversario()
    { ... }
    public float getTxRef()
    { ... }
    public void correcao()
    { ... }
    public void extrato()
    { ... }
}
```



**Herança**

# Herança

- Derivando Classes em JAVA

```
class ContaPoupanca extends Conta  
{
```

```
    private Data aniversario;  
    private float txRef;
```



Atributos específicos da Subclasse

```
    public ContaPoupanca(String cpf, String nome, int d, int m, int a)  
    { ... }  
    public void setAniversario(int d, int m, int a)  
    { ... }  
    public void setAniversario(Data dt)  
    { ... }  
    public void setTxRef(float tx)  
    { ... }  
    public Data getAniversario()  
    { ... }  
    public float getTxRef()  
    { ... }  
    public void correcao()  
    { ... }  
    public void extrato()  
    { ... }  
}
```

# Herança

- Derivando Classes em JAVA

Atributos [private]  
herdados

String numero;  
String titular;  
String cpf;  
boolean bloqueada;  
float saldo;

```
class ContaPoupanca extends Conta  
{
```

```
    private Data aniversario;  
    private float txRef;
```



Atributos específicos da Subclasse


```
    public ContaPoupanca(String cpf, String nome, int d, int m, int a)  
    { ... }  
    public void setAniversario(int d, int m, int a)  
    { ... }  
    public void setAniversario(Data dt)  
    { ... }  
    public void setTxRef(float tx)  
    { ... }  
    public Data getAniversario()  
    { ... }  
    public float getTxRef()  
    { ... }  
    public void correcao()  
    { ... }  
    public void extrato()  
    { ... }  
}
```

# Herança

- Derivando Classes em JAVA

```
class ContaPoupanca extends Conta
{
    private Data aniversario;
    private float txRef;

    public ContaPoupanca(String cpf, String nome, int d, int m, int a)
    { ... }
    public void setAniversario(int d, int m, int a)
    { ... }
    public void setAniversario(Data dt)
    { ... }
    public void setTxRef(float tx)
    { ... }
    public Data getAniversario()
    { ... }
    public float getTxRef()
    { ... }
    public void correcao()
    { ... }
    public void extrato()
    { ... }
}
```



Métodos  
específicos  
da  
Subclasse

Métodos [public]  
herdados

```
Conta();
Conta(String, String);
void setNumero(String);
void setTitular(String);
void setCpf(String);
void setBloqueada(boolean);
void setSaldo(float);
String getNumero();
String getTitular();
String getCpf();
boolean isBloqueada();
float getSaldo();
boolean saque(float);
boolean deposito(float);
void extrato();
static String geraNumero(int);
```




# Herança

- Derivando Classes em JAVA

```
class ContaPoupanca extends Conta
{
    private Data aniversario;
    private float txRef;

    public ContaPoupanca(String cpf, String nome, int d, int m, int a)
    { ... }
    public void setAniversario(int d, int m, int a)
    { ... }
    public void setAniversario(Data dt)
    { ... }
    public void setTxRef(float tx)
    { ... }
    public Data getAniversario()
    { ... }
    public float getTxRef()
    { ... }
    public void correcao()
    { ... }
    public void extrato()
    { ... }
}
```



Métodos  
específicos  
da  
Subclasse

Métodos [public]  
herdados

**Redefinição de Método**

```
Conta();
Conta(String, String);
void setNumero(String);
void setTitular(String);
void setCpf(String);
void setBloqueada(boolean);
void setSaldo(float);
String getNumero();
String getTitular();
String getCpf();
boolean isBloqueada();
float getSaldo();
boolean saque(float);
boolean deposito(float);
void extrato();
static String geraNumero(int);
```

# Herança

- Derivando Classes em JAVA

```
public void extrato()  
{  
    System.out.println("=====");  
    System.out.println("CONTA : " + this.getNumero());  
    System.out.println("CPF ..: " + this.getCpf());  
    System.out.println("NOME ..: " + this.getTitular());  
    System.out.println("SALDO : R$" + this.getSaldo());  
    System.out.println("=====");  
}
```

Invocando o método extrato() da classe base.

```
public void extrato()  
{  
    super.extrato();  
    System.out.println("ANIVERSARIO: dia " + this.getAniversario().getDia());  
    System.out.println("=====");  
}
```

# Herança

- Construtores da Classe Derivada em JAVA
  - Quando a subclasse não define um construtor, o construtor correspondente da superclasse é invocado.
  - Caso deseje invocar um construtor da superclasse do construtor da subclasse, é possível usando a palavra reservada `super()`, com a lista de argumentos do construtor desejado.

```
public ContaPoupanca(String cpf, String nome, int d, int m, int a)
{
    super(cpf, nome);
    this.setAniversario(d, m, a);
}
```

- Neste construtor da subclasse `ContaPoupanca`, o construtor `Conta(String, String)` da superclasse `Conta` é invocado, e além disso, o atributo específico aniversário da subclasse é inicializado.

# Herança

- Construtores da Classe Derivada em JAVA
  - Quando a subclasse não define um construtor, o construtor correspondente da superclasse é invocado.
  - Caso deseje invocar um construtor da superclasse do construtor da subclasse, é possível usando a palavra reservada `super()`, com a lista de argumentos do construtor desejado.

```
public ContaPoupanca(String cpf, String nome, int d, int m, int a)
{
    super(cpf, nome);
    this.setAniversario(d, m, a);
}
```

← **Invocando o construtor da classe base.**

- Neste construtor da subclasse `ContaPoupanca`, o construtor `Conta(String, String)` da superclasse `Conta` é invocado, e além disso, o atributo específico aniversário da subclasse é inicializado.

# Herança

- Derivando Classes em JAVA

```
public class Main
{
    public static void main(String[] args)
    {
        ContaPoupanca cta = new ContaPoupanca("22388645633", "Monael", 05, 10, 2016);
        cta.extrato();
        cta.deposito(100);
        cta.extrato();
    }
}
```

# Herança

- Derivando Classes em JAVA

```
public class Main
{
    public static void main(String[] args)
    {
        ContaPoupanca cta = new ContaPoupanca("22388645633", "Monael", 05, 10, 2016);
        cta.extrato();
        cta.deposito(100);
        cta.extrato();
    }
}
```

```
=====
CONTA : 8620019005
CPF ..: 22388645633
NOME ..: Monael
SALDO : R$0
=====
ANIVERSARIO: dia 5
=====
```

# Herança

- Derivando Classes em JAVA

```
public class Main
{
    public static void main(String[] args)
    {
        ContaPoupanca cta = new ContaPoupanca("22388645633", "Monael", 05, 10, 2016);
        cta.extrato();
        cta.deposito(100);
        cta.extrato();
    }
}
```

**Invocando o método depósito(float)  
que fora definido na classe base.**

```
=====
CONTA : 8620019005
CPF ..: 22388645633
NOME ..: Monael
SALDO : R$100
=====
ANIVERSARIO: dia 5
=====
```

# Herança

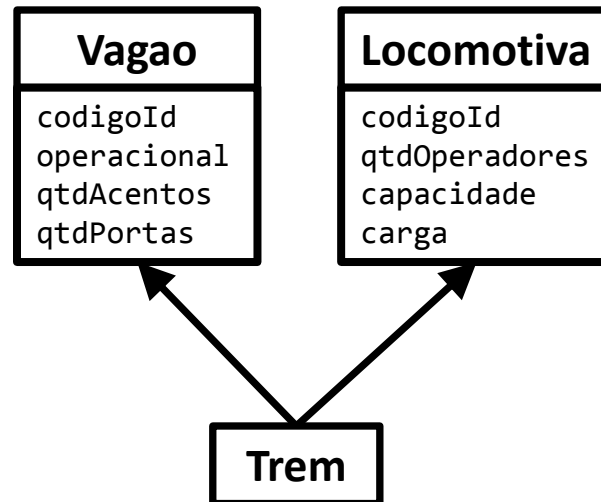
- Casting entre Classe Pai e Classe Filho
  - A conversão entre superclasse e subclasse é implícita.
  - Todo objeto da subclasse é implicitamente convertido para objeto da superclasse.
  - Contudo, a recíproca não é verdadeira.

```
int main()
{
    Conta *agencia[10];
    int i;
    std::string cpf, nome;
    for(i=0; i<10; i++)
    {
        std::cin >> cpf >> nome;
        agencia[i] = new ContaPoupanca(cpf, nome, 10, 10, 2014);
    }
    return 0;
}
```



# Herança

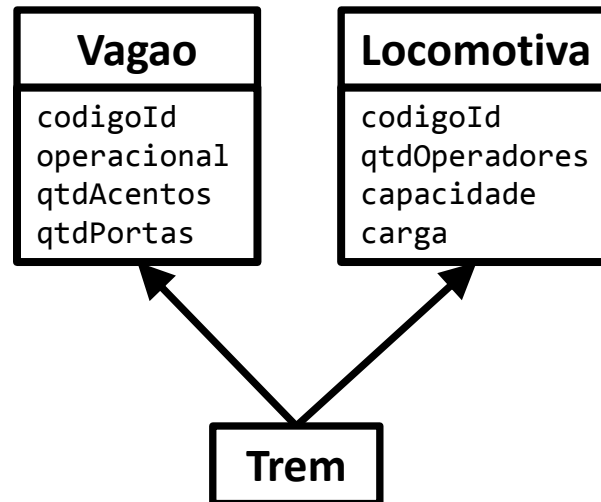
- Herança Múltipla
  - Quando uma classe herda características de mais de uma classe base, dá-se o nome de Herança Múltipla.



- Perceba que neste caso, um objeto da classe Trem herda os atributos e métodos de Locomotiva e de Vagão, porém um trem tem vários vagões e não é possível “herdar várias vezes” da classe vagão.

# Herança

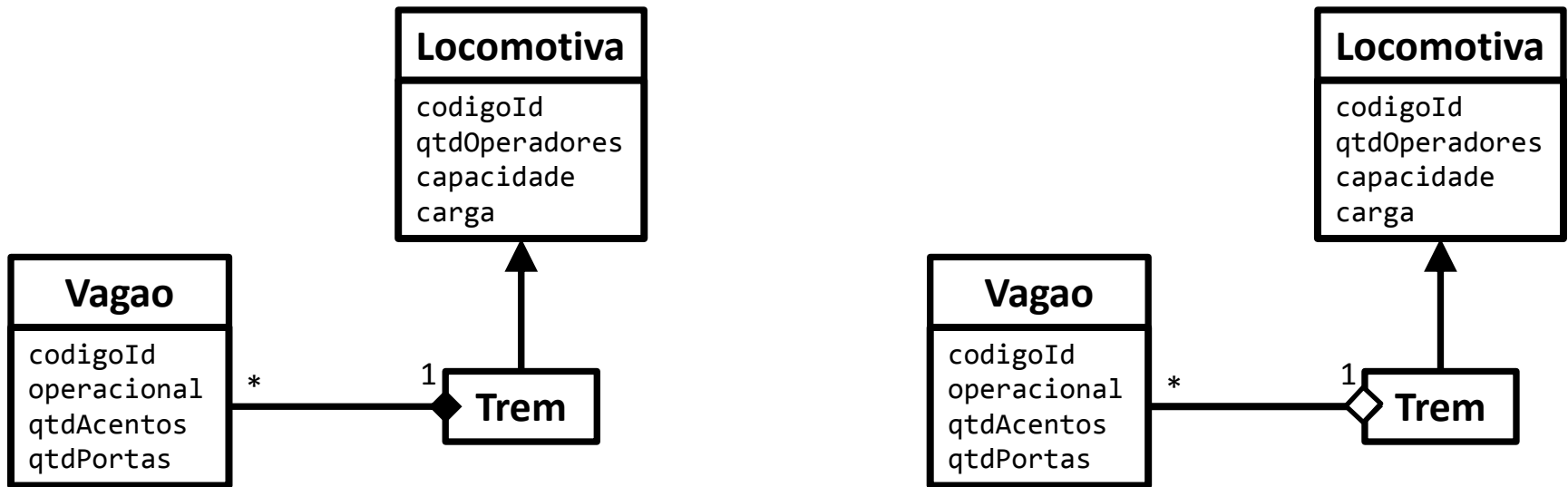
- Herança Múltipla
  - Quando uma classe herda características de mais de uma classe base, dá-se o nome de Herança Múltipla.



- Neste caso a herança não é a melhor forma de modelar o relacionamento das classes Trem e Vagão.
- Seria através de uma **Agregação** ou **Composição**.

# Herança

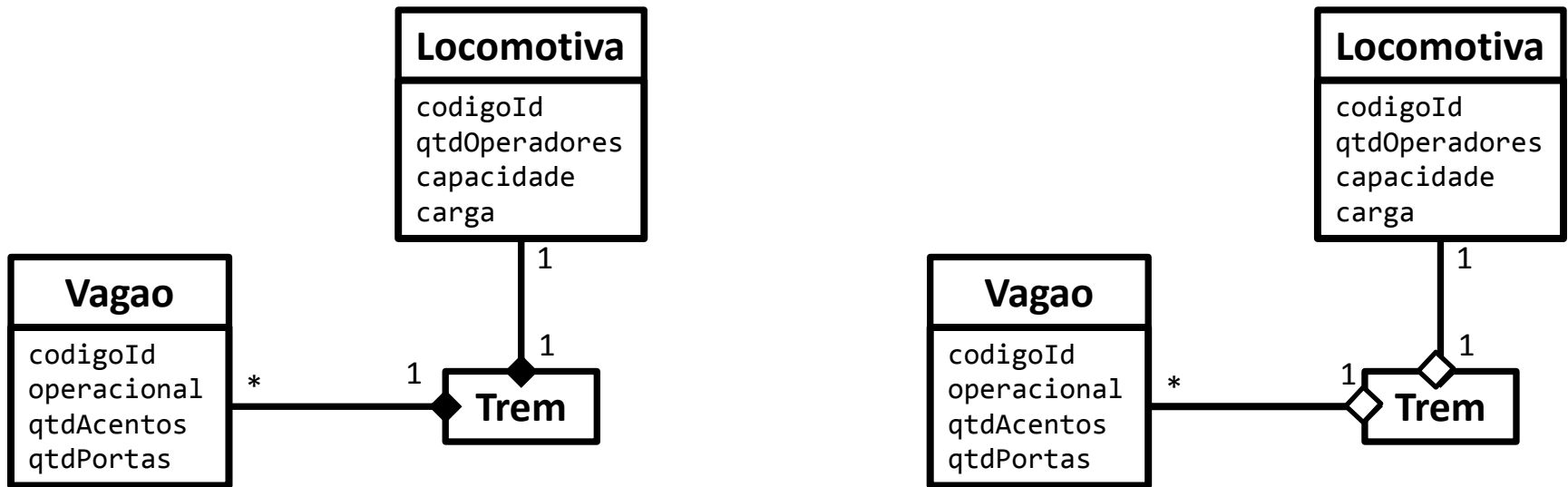
- Herança Múltipla
  - Quando uma classe herda características de mais de uma classe base, dá-se o nome de Herança Múltipla.



- Neste caso a herança não é a melhor forma de modelar o relacionamento das classes Trem e Vagão.
- Seria através de uma **Agregação** ou **Composição**.

# Herança

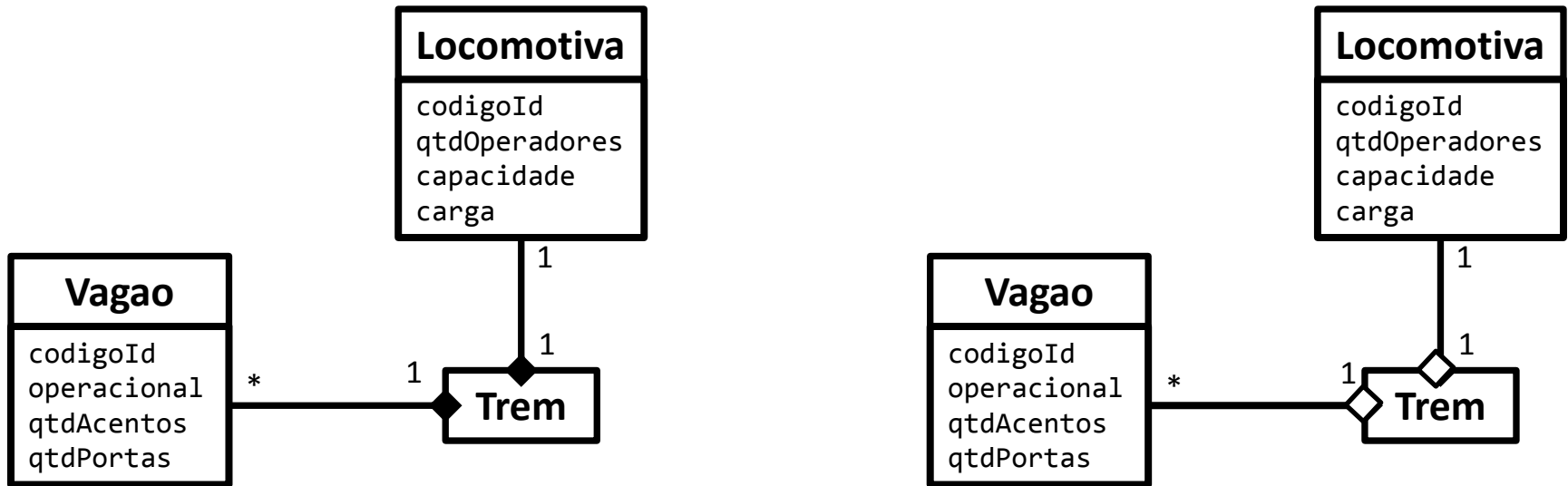
- Herança Múltipla
  - Quando uma classe herda características de mais de uma classe base, dá-se o nome de Herança Múltipla.



- Neste caso a herança não é a melhor forma de modelar o relacionamento das classes Trem e Vagão.
- Seria através de uma **Agregação** ou **Composição**.

# Herança

- Herança Múltipla
  - Quando uma classe herda características de mais de uma classe base, dá-se o nome de Herança Múltipla.

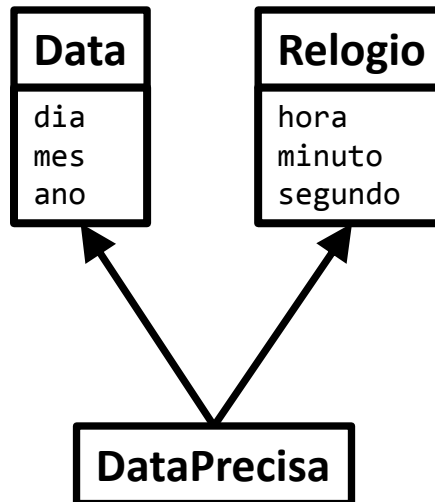


- Neste caso a herança não é a melhor forma de modelar o relacionamento das classes Trem e Vagão.
- Seria através de uma **Agregação** ou **Composição**.

**Tudo depende do  
Problema que você  
está modelando.**

# Herança

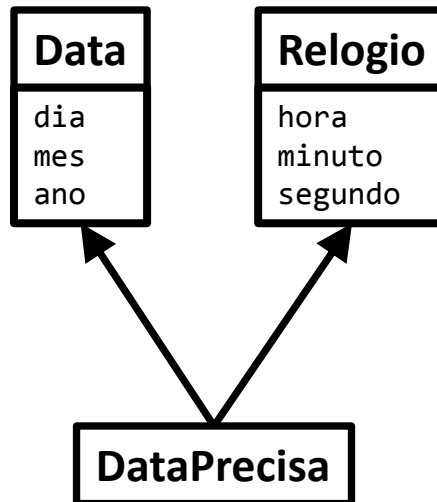
- Herança Múltipla
  - Quando uma classe herda características de mais de uma classe base, dá-se o nome de Herança Múltipla.



- Exemplo: Definir uma Classe **DataPrecisa**, que armazena dia, mês, ano, hora, minuto e segundo.

# Herança

- Herança Múltipla
  - Quando uma classe herda características de mais de uma classe base, dá-se o nome de Herança Múltipla.



- Perceba que neste caso, um objeto da classe **DataPrecisa** terá os mesmos atributos e métodos das classes **Data** e **Relogio** adicionais aos seus atributos e métodos específicos.

# Herança

- Herança Múltipla em C++

```
class Data
{
    private:
        int dia, mes, ano;
    public:
        Data();
        Data(int, int, int);
        void setDia(int);
        void setMes(int);
        void setAno(int);
        int getDia();
        int getMes();
        int getAno();
        bool valida();
        bool bissexto();
        static int ultimoDiaDoMes(Data);
        void incrementa();
        void decrementa();
};
```

```
class Relogio
{
    private:
        int hora, minuto, segundo;
    public:
        Relogio();
        Relogio(int, int, int);
        void setHora(int);
        void setMinuto(int);
        void setSegundo(int);
        int getHora();
        int getMinuto();
        int getSegundo();
        void incrementa();
        void decrementa();
};
```



# Herança

- Herança Múltipla em C++

```
class DataPrecisa : public Relogio, public Data
{
    private:
        bool regressivo;
    public:
        DataPrecisa();
        DataPrecisa(int, int, int, int, int, int);
        DataPrecisa(Data);
        DataPrecisa(Data, Relogio);
        void setRegressivo(bool);
        bool isRegressivo();
        DataPrecisa operator ++(int);
        DataPrecisa operator --(int);
};
```

# Herança

- Construtores na Herança Múltipla
  - Quando a classe derivada não tem construtor definido, então os construtores vazios das classes base são invocados.
  - Quando a classe derivada define um construtor, então os construtores vazios das classes base são invocados, e na sequência o seu próprio construtor é invocado.
  - Caso deseje que outros construtores (não vazios) sejam invocados, deve-se explicitar suas invocações como na herança simples.
  - Use vírgula para cada invocação individual.

# Herança

- Herança Múltipla em C++

```
DataPrecisa::DataPrecisa(int d, int m, int a, int h, int mi, int s)
{
    this->setDia(d);
    this->setMes(m);
    this->setAno(a);
    this->setHora(h);
    this->setMinuto(mi);
    this->setSegundo(s);
    this->setRegressivo(false);
}
```

```
DataPrecisa::DataPrecisa(Data dt) : Data(dt.getDia(), dt.getMes(), dt.getAno())
{
    this->setHora(0);
    this->setMinuto(0);
    this->setSegundo(0);
    this->setRegressivo(false);
}
```

# Herança

- Herança Múltipla em C++

```
DataPrecisa::DataPrecisa(Data dt, Relogio rel) :  
    Data(dt.getDia(), dt.getMes(), dt.getAno()),  
    Relogio(rel.getHora(), rel.getMinuto(), rel.getSegundo())  
{  
    this->setRegressivo(false);  
}
```

# Herança

- Herança Múltipla em C++

```
DataPrecisa DataPrecisa::operator ++(int)
{
    DataPrecisa dtp(
        Data(this->getDia(), this->getMes(), this->getAno()),
        Relogio(this->getHora(), this->getMinuto(), this->getSegundo())
    );
    this->incrementa();
    if(this->getHora() > 23)
    {
        this->setHora(0);
        this->incrementa();
    }
    return dtp;
}
```

# Herança

- Herança Múltipla em C++

```
DataPrecisa DataPrecisa::operator ++(int)
{
    DataPrecisa dtp(
        Data(this->getDia(), this->getMes(), this->getAno()),
        Relogio(this->getHora(), this->getMinuto(), this->getSegundo())
    );
    this->incrementa();
    if(this->getHora() > 23)
    {
        this->setHora(0);
        this->incrementa();
    }
    return dtp;
}
```

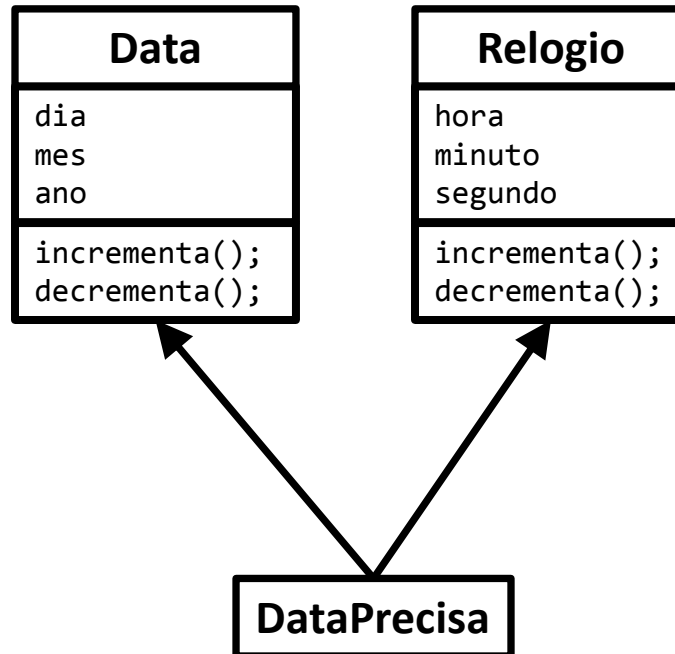
```
DataPrecisa.cpp: In member function 'DataPrecisa DataPrecisa::operator++(int)':
DataPrecisa.cpp:99:11: error: request for member 'incrementa' is ambiguous
    this->incrementa();
    ^
```

```
DataPrecisa.cpp:38:18: note: candidates are: void Data::incrementa()
    void incrementa();
    ^
```

```
DataPrecisa.cpp:16:18: note: void Relogio::incrementa()
    void incrementa();
    ^
```

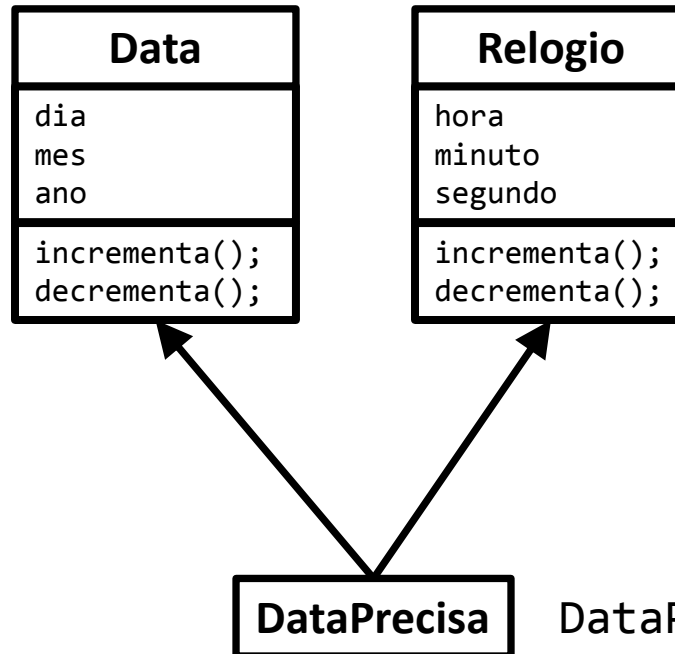
# Herança

- Ambiguidade na Herança Múltipla
  - Ocorre uma ambiguidade em herança múltipla quando nas classes base há métodos ou atributos com o mesmo nome, e na classe derivada não há redefinição.
  - Neste caso, ao invocar o método ou acessar o atributo não se sabe qual deve-se considerar.



# Herança

- Ambiguidade na Herança Múltipla
  - Ocorre uma ambiguidade em herança múltipla quando nas classes base há métodos ou atributos com o mesmo nome, e na classe derivada não há redefinição.
  - Neste caso, ao invocar o método ou acessar o atributo não se sabe qual deve-se considerar.

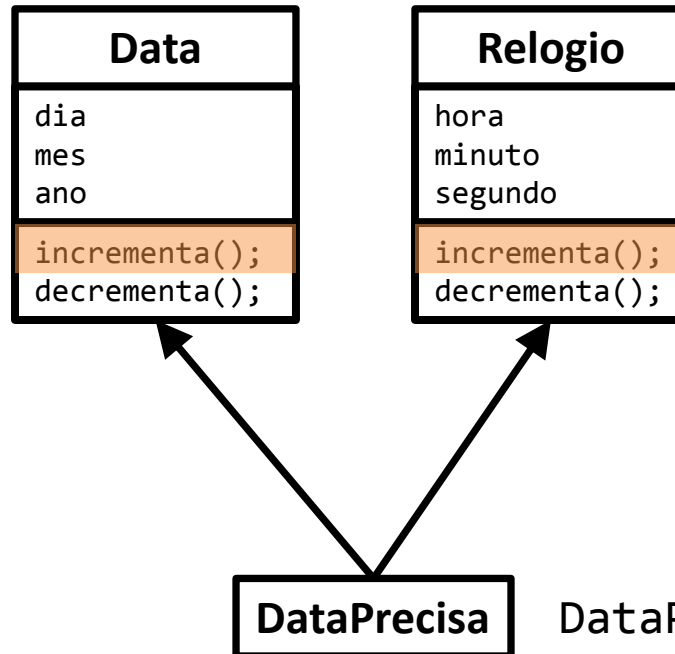


`DataPrecisa:obj.incrementa();`



# Herança

- Ambiguidade na Herança Múltipla
  - Ocorre uma ambiguidade em herança múltipla quando nas classes base há métodos ou atributos com o mesmo nome, e na classe derivada não há redefinição.
  - Neste caso, ao invocar o método ou acessar o atributo não se sabe qual deve-se considerar.



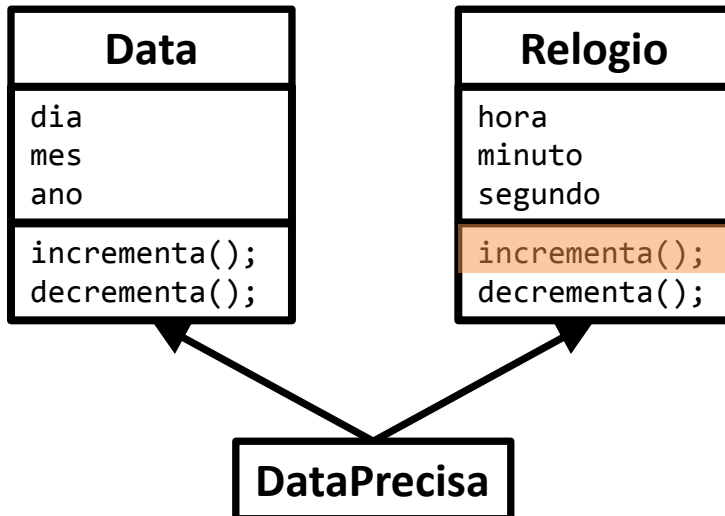
DataPrecisa:obj.incrementa();

# Herança

- Ambiguidade na Herança Múltipla
  - Ocorre uma ambiguidade em herança múltipla quando nas classes base há métodos ou atributos com o mesmo nome, e na classe derivada não há redefinição.
  - Neste caso, ao invocar o método ou acessar o atributo não se sabe qual deve-se considerar.
  - Assim, deve-se informar ao compilador qual é a Classe Base de origem da chamada ambígua, para que ele saiba resolver.
  - Para isso, usa-se o operador de resolução de escopo (::).

# Herança

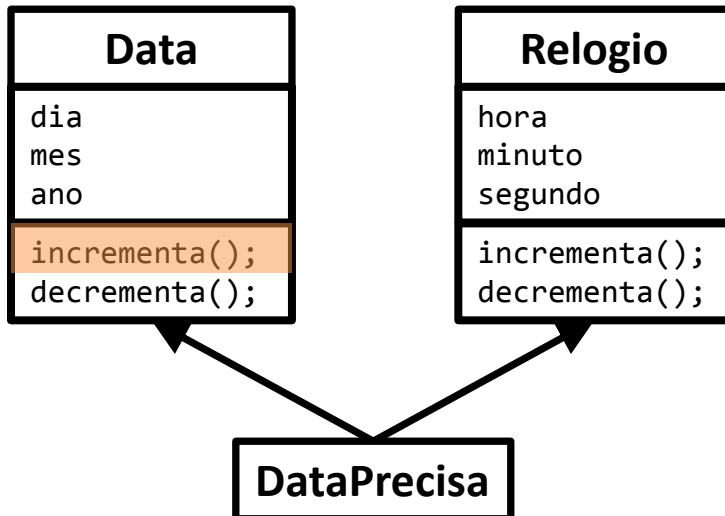
- Ambiguidade na Herança Múltipla
  - Ocorre uma ambiguidade em herança múltipla quando nas classes base há métodos ou atributos com o mesmo nome, e na classe derivada não há redefinição.
  - Neste caso, ao invocar o método ou acessar o atributo não se sabe qual deve-se considerar.
  - Assim, deve-se informar ao compilador qual é a Classe Base de origem da chamada ambígua, para que ele saiba resolver.
  - Para isso, usa-se o operador de resolução de escopo (::).



`obj.Relogio::incrementa();`

# Herança

- Ambiguidade na Herança Múltipla
  - Ocorre uma ambiguidade em herança múltipla quando nas classes base há métodos ou atributos com o mesmo nome, e na classe derivada não há redefinição.
  - Neste caso, ao invocar o método ou acessar o atributo não se sabe qual deve-se considerar.
  - Assim, deve-se informar ao compilador qual é a Classe Base de origem da chamada ambígua, para que ele saiba resolver.
  - Para isso, usa-se o operador de resolução de escopo (::).



`obj.Data::incrementa();`

# Herança

- Herança Múltipla em C++

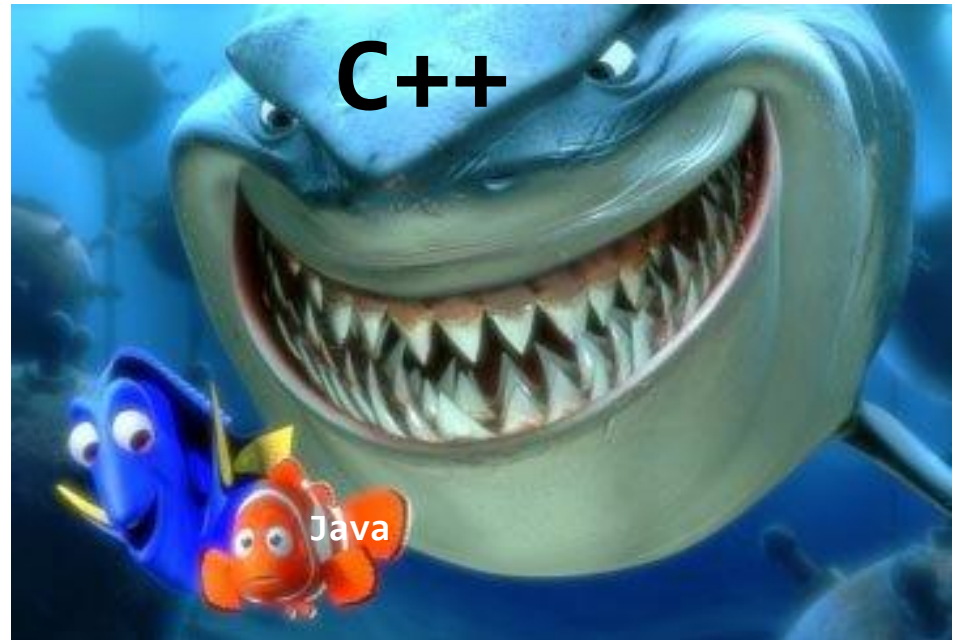
```
DataPrecisa DataPrecisa::operator ++(int)
{
    DataPrecisa dtp(
        Data(this->getDia(), this->getMes(), this->getAno()),
        Relogio(this->getHora(), this->getMinuto(), this->getSegundo())
    );
    this->Relogio::incrementa();
    if(this->getHora() > 23)
    {
        this->setHora(0);
        this->Data::incrementa();
    }
    return dtp;
}
```

# Herança

- Herança Múltipla em JAVA

# Herança

- Herança Múltipla em JAVA
  - JAVA não permite heranças múltiplas.
  - Em JAVA só é possível fazer herança simples.
  - Um mecanismo para “emular” heranças múltiplas em JAVA é o uso das interfaces.



# Interfaces

- Com intuito de prevenir ambiguidades e tornar o passeio pela hierarquia de classes mais eficiente, a Linguagem JAVA não permite heranças múltiplas.
- Contudo, JAVA oferece um mecanismo que simula heranças múltiplas chamado interfaces.
- Uma interface é definida como uma classe, porém sem a implementação dos seus métodos.
- Quando uma classe implementa uma interface é como um “contrato” realizado entre a classe a interface, onde a classe se compromete a implementar todos os métodos daquela interface.



# Interfaces

