



Universidade Federal do ABC  
Centro de Matemática, Computação e Cognição

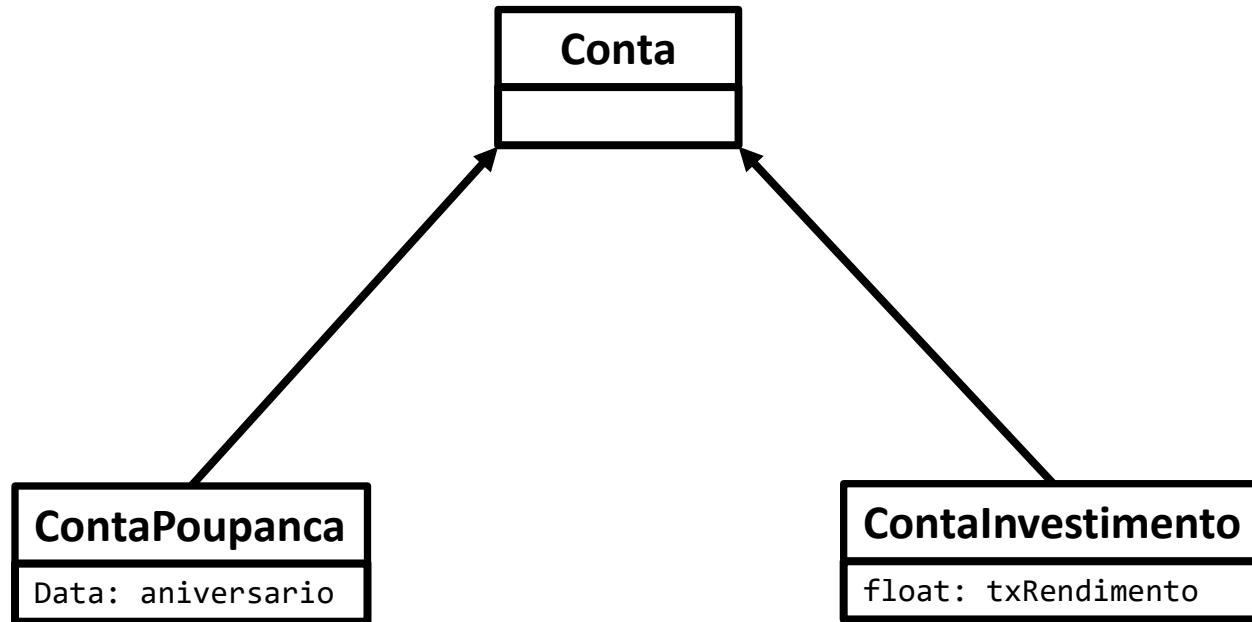
# Programação Orientada a Objetos

Monael Pinheiro Ribeiro, D.Sc.

# Polimorfismo

- Invocar um método de um objeto, sem especificar o tipo exato do objeto também é um tipo de chamada polimórfica.
- Polimorfismo é a capacidade de uma única instrução invocar diferentes métodos e assumir formas diferentes.
- Com o mecanismo de Herança chamadas polimórficas são consequência das derivações promovidas.

# Polimorfismo



# Classe Conta

- 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);
};

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;
}

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

# Classe ContaPoupanca

- Derivando Classes em C++ (Classe Derivada)

```
class ContaPoupanca : public Conta
{
```

```
    private:
```

```
        Data aniversario;
```

```
    public:
```

```
        ContaPoupanca(std::string, std::string, int, int, int);
```

```
        void setAniversario(int, int, int);
```

```
        void setAniversario(Data);
```

```
        Data getAniversario();
```

```
        void correcao();
```

```
        void extrato();
```

```
};
```

```
void ContaPoupanca::extrato()
```

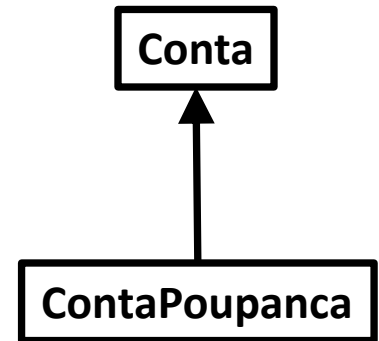
```
{
```

```
    Conta::extrato();
```

```
    std::cout << "ANIVERSARIO: dia " << this->getAniversario().getDia() << std::endl;
```

```
    std::cout << "===== " << std::endl;
```

```
}
```



```
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();
```

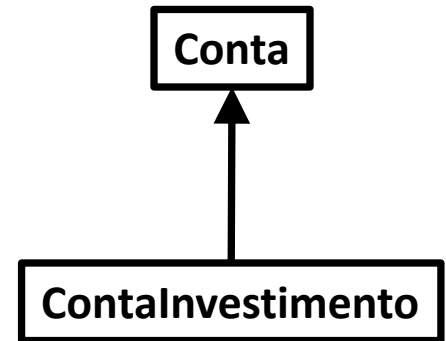
```
};
```

# Classe ContaInvestimento

- Derivando Classes em C++ (Classe Derivada)

```
class ContaInvestimento : public Conta
{
    private:
        float txRendimento;
        float txAdministracao;
    public:
        ContaInvestimento(std::string, std::string, float);
        void setTxRendimento(float);
        float getTxRendimento();
        void setTxAdministracao(float);
        float getTxAdministracao();
        void correcao();
        void extrato();
};

void ContaInvestimento::extrato()
{
    Conta::extrato();
    std::cout << " TAXA DE RENDIMENTO: " << getTxRendimento() << "\%.m" << std::endl;
    std::cout << " TAXA DE ADMINISTRACAO: " << getTxAdministracao() << "\%.m" << std::endl;
    std::cout << "===== " << std::endl;
}
```

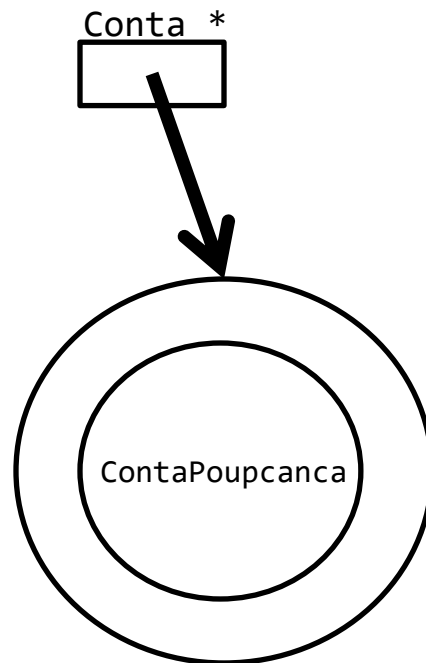


# Ponteiro para Classe-Base

- Ao alocar um ponteiro para classe-base, você pode instanciar objetos para quaisquer uma de suas classes derivadas.

# Ponteiro para Classe-Base

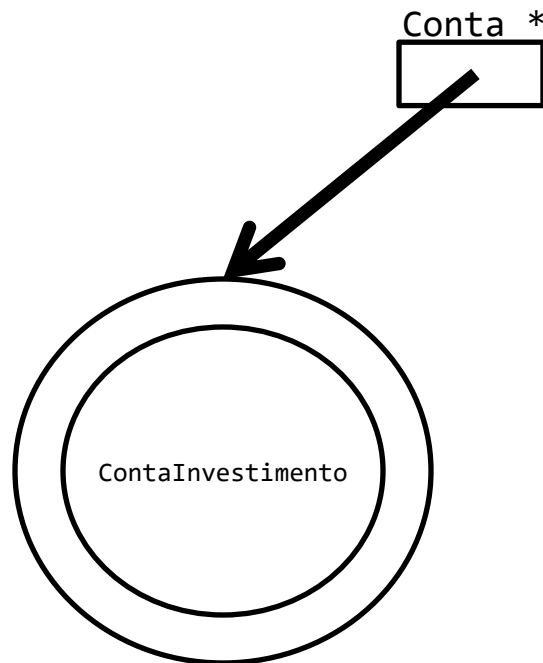
- Ao alocar um ponteiro para classe-base, você pode instanciar objetos para quaisquer uma de suas classes derivadas.





# Ponteiro para Classe-Base

- Ao alocar um ponteiro para classe-base, você pode instanciar objetos para quaisquer uma de suas classes derivadas.



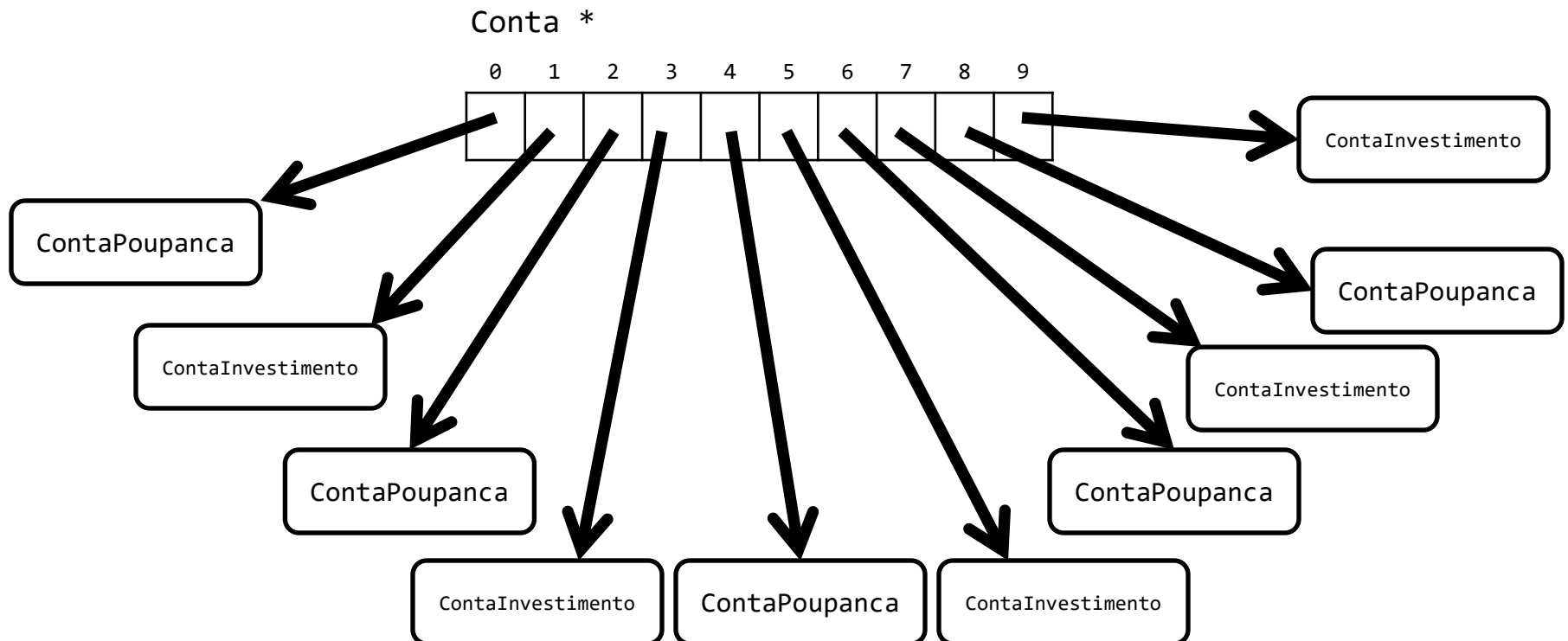
# Ponteiro para Classe-Base

- Ao alocar um ponteiro para classe-base, você pode instanciar objetos para quaisquer uma de suas classes derivadas.

[illegible]

# Ponteiro para Classe-Base

- Ao alocar um ponteiro para classe-base, você pode instanciar objetos para quaisquer uma de suas classes derivadas.



# Ponteiro para Classe-Base

```
int main
{
    Conta *agencia[10];
    int i;

    for(i=0; i<10; i++)
    {
        if(i%2==0) agencia[i] = new ContaPoupanca("123", "Fulano", 12, 7, 2016);
        else agencia[i] = new ContaInvestimento("456", "Beltrano", 10);
        agencia[i]->deposito(100);
    }

    for(i=0; i<10; i++)
    {
        agencia[i]->extrato();
    }
    return 0;
}
```

# Ponteiro para Classe-Base

```
=====
CONTA : 0294804783
CPF ..: 123
NOME ..: Fulano
SALDO : R$100
=====

CONTA : 0294804783
CPF ..: 456
NOME ..: Beltrano
SALDO : R$100
=====

...

=====
CONTA : 0294804783
CPF ..: 123
NOME ..: Fulano
SALDO : R$100
=====

CONTA : 0294804783
CPF ..: 456
NOME ..: Beltrano
SALDO : R$100
=====
```

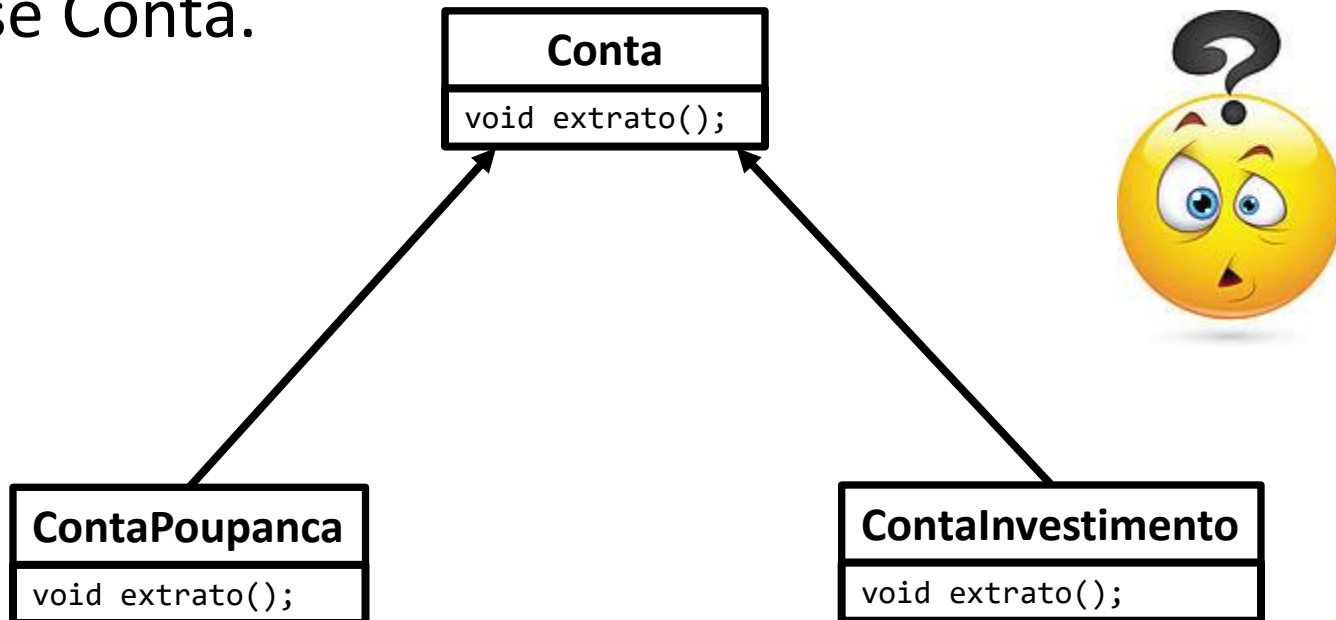
# Ponteiro para Classe-Base

```
=====
CONTA : 0294804783
CPF ..: 123
NOME ..: Fulano
SALDO : R$100
=====
CONTA : 0294804783
CPF ..: 456
NOME ..: Beltrano
SALDO : R$100
=====
...
=====
CONTA : 0294804783
CPF ..: 123
NOME ..: Fulano
SALDO : R$100
=====
CONTA : 0294804783
CPF ..: 456
NOME ..: Beltrano
SALDO : R$100
=====
```



# Ponteiro para Classe-Base

- Note que embora temos um vetor de ponteiros `Conta`, e alocado no vetor objetos do tipo `ContaPoupanca` e `ContaInvestimento` o método `extrato()` invocado foi o método original da classe `Conta`.



# Resolução Absoluta de Métodos

- A resolução da chamada ao método é feita em tempo de compilação

```
int main
{
    Conta *agencia[10];
    int i;

    for(i=0; i<10; i++)
    {
        if(i%2==0) agencia[i] = new ContaPoupanca("123", "Fulano", 12, 7, 2016);
        else agencia[i] = new ContaInvestimento("456", "Beltrano", 10);
        agencia[i]->deposito(100);
    }

    for(i=0; i<10; i++)
    {
        agencia[i]->extrato();
    }
    return 0;
}
```



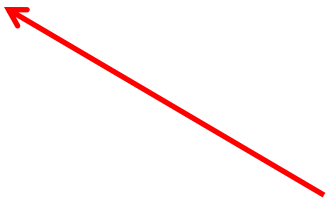
# Resolução Absoluta de Métodos

- A resolução da chamada ao método é feita em tempo de compilação

```
int main
{
    Conta *agencia[10];
    int i;

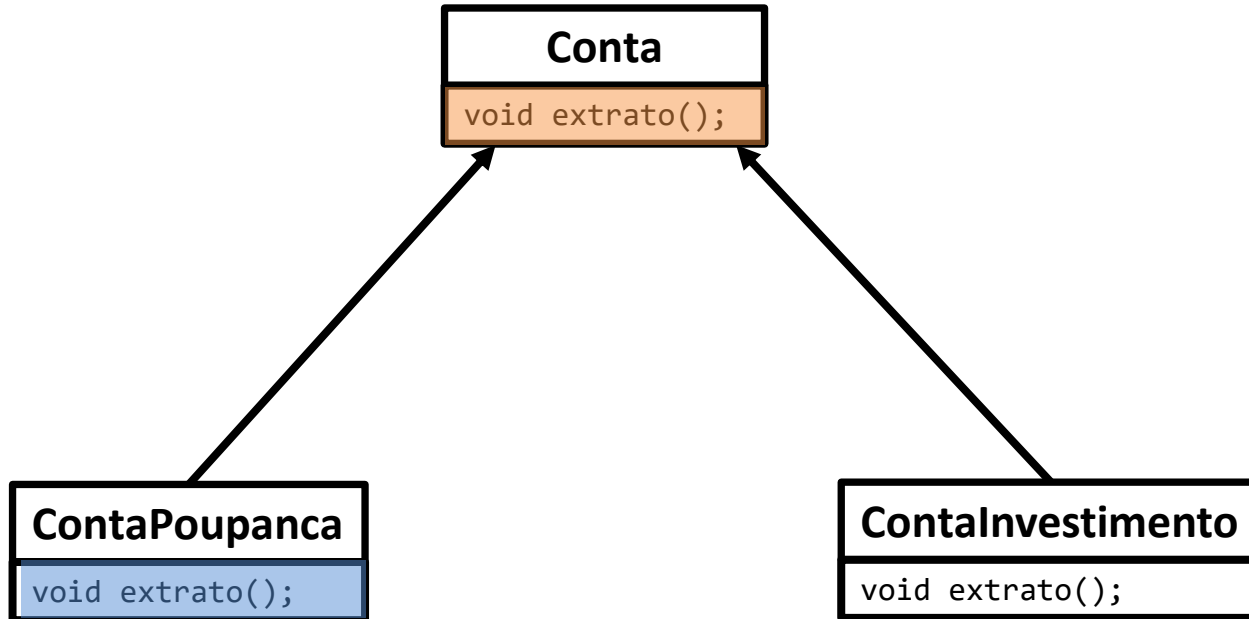
    for(i=0; i<10; i++)
    {
        if(i%2==0) agencia[i] = new ContaPoupanca("123", "Fulano", 12, 7, 2016);
        else agencia[i] = new ContaInvestimento("456", "Beltrano", 10);
        agencia[i]->deposito(100);
    }

    for(i=0; i<10; i++)
    {
        agencia[i]->extrato();
    }
    return 0;
}
```



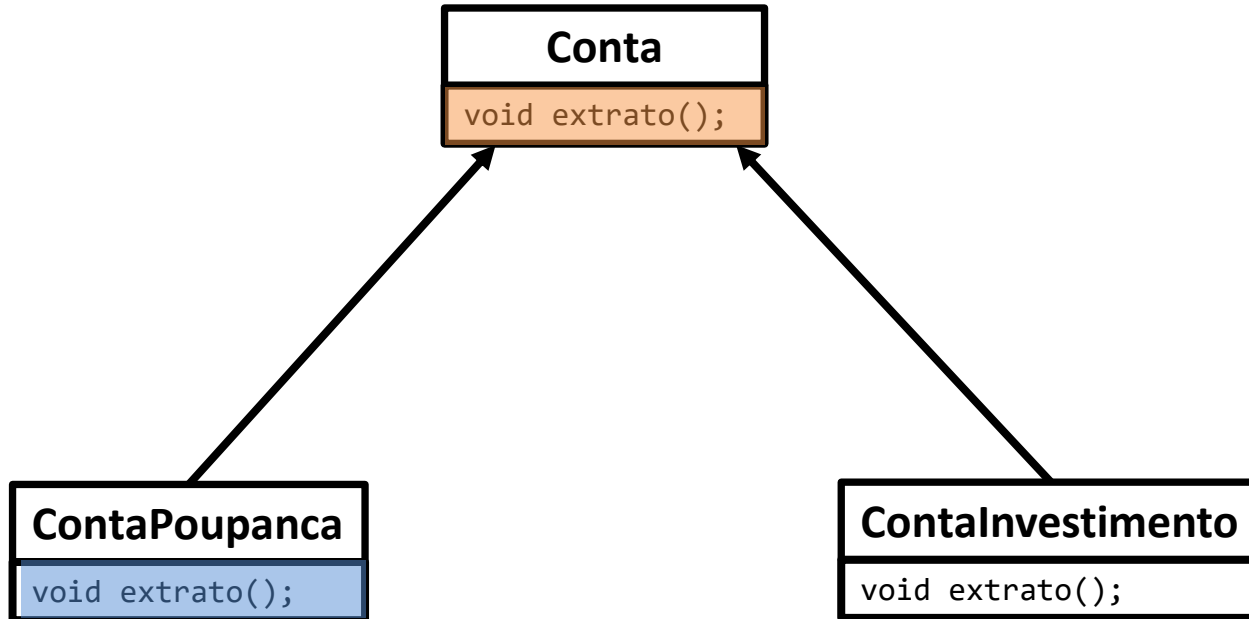
Não se sabe a priori, para objeto de qual tipo aponta **agencia[i]**

# Resolução Absoluta de Métodos



```
ContaPoupanca objCtaPoup("123", "Fulano", 12, 7, 2016);
Conta *objConta = &objCtaPoup;
objConta->extrato();
objCtaPoup.extrato();
```

# Resolução Absoluta de Métodos



```
ContaPoupanca objCtaPoup("123", "Fulano", 12, 7, 2016);
Conta *objConta = &objCtaPoup;
objConta->extrato();
objCtaPoup.extrato();
```

```
=====
CONTA : 3408387998
CPF ..: 123
NOME ..: Fulano
SALDO : R$0
=====
```

```
CONTA : 3408387998
CPF ..: 123
NOME ..: Fulano
SALDO : R$0
=====
```

```
ANIVERSARIO: dia 12
=====
```

# Métodos Virtuais

- Um método virtual faz-se necessário quando invoca-se um método específico de uma classe derivada, por meio de um ponteiro para a sua classe-base.
- Um método na classe-base redefinido na classe derivada, e se referencia ao objeto da classe derivada através de um ponteiro para classe-base, o método da classe-base será invocado.
- Exceto, se o método for declarado como virtual na classe-base.

# Métodos Virtuais

- Em outras palavras ...
- A chamada a um método virtual é resolvido em **tempo de execução**.
- Com isso, é possível averiguar qual o tipo de dado exato está alocado naquele momento e invocar o método preciso daquele tipo.
- A essa forma de resolver o método se chama **Resolução Dinâmica ou Tardia**.

# Classe Conta

- 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);
        virtual void extrato();
        static std::string geraNumero(int);
};

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

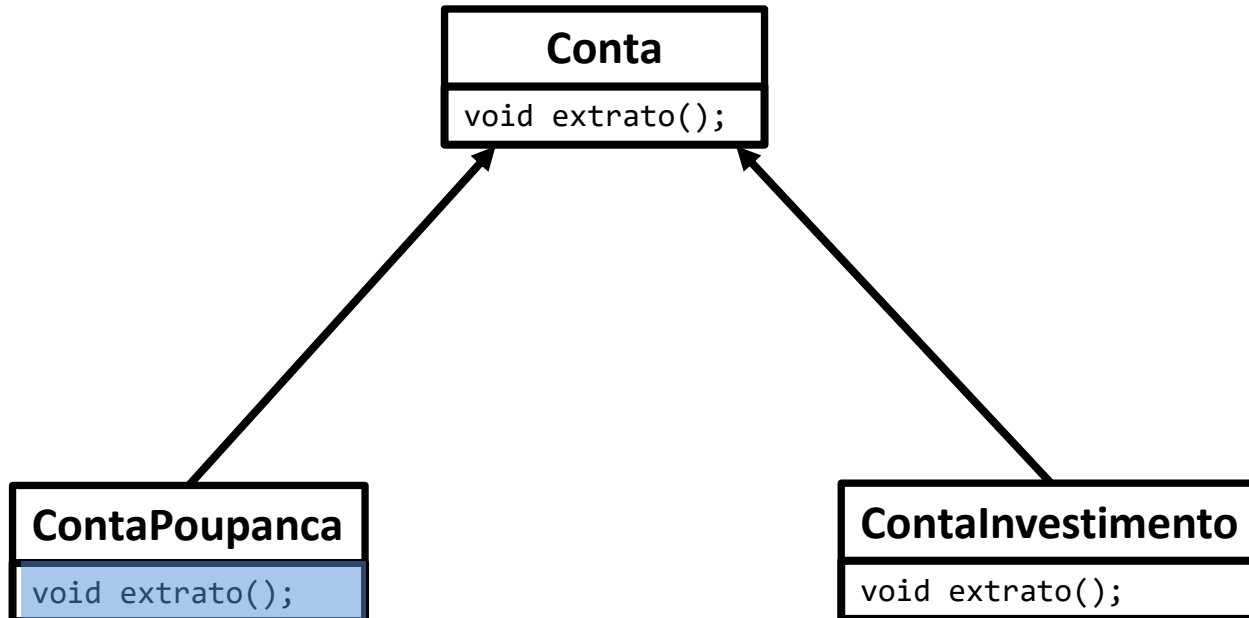
# Classe Conta

- 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);
        virtual void extrato(); ← Método Virtual
        static std::string geraNumero(int);
};

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

# Métodos Virtuais



```
ContaPoupanca objCtaPoup("123", "Fulano", 12, 7, 2016);
Conta *objConta = &objCtaPoup;
objConta->extrato();
objCtaPoup.extrato();
```

```
=====
CONTA : 3408387998
CPF ..: 123
NOME ..: Fulano
SALDO : R$0
```

```
=====
ANIVERSARIO: dia 12
=====
```

```
CONTA : 3408387998
CPF ..: 123
NOME ..: Fulano
SALDO : R$0
```

```
=====
ANIVERSARIO: dia 12
=====
```



# Métodos Virtuais

```
int main
{
    Conta *agencia[10];
    int i;

    for(i=0; i<10; i++)
    {
        if(i%2==0) agencia[i] = new ContaPoupanca("123", "Fulano", 12, 7, 2016);
        else agencia[i] = new ContaInvestimento("456", "Beltrano", 10);
        agencia[i]->deposito(100);
    }

    for(i=0; i<10; i++)
    {
        agencia[i]->extrato();
    }
    return 0;
}
```

# Métodos Virtuais

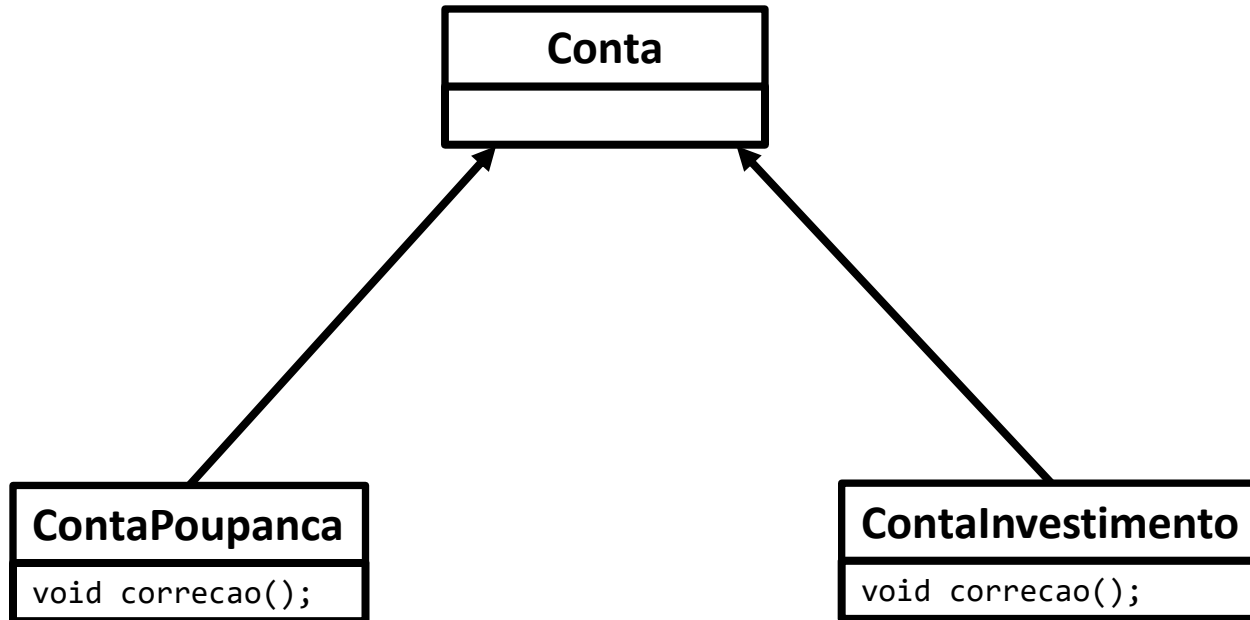
```
int main
{
    Conta *agencia[10];
    int i;

    for(i=0; i<10; i++)
    {
        if(i%2==0) agencia[i] = new ContaPoupanca("123", "Fulano", 12, 7, 2016);
        else agencia[i] = new ContaInvestimento("456", "Beltrano", 10);
        agencia[i]->deposito(100);
    }

    for(i=0; i<10; i++)
    {
        agencia[i]->extrato();
    }
    return 0;
}
```

```
=====
CONTA : 0294804783
CPF ..: 123
NOME ..: Fulano
SALDO :R$100
=====
TAXA DE RENDIMENTO: 10% a.m
TAXA DE ADMINISTRACAO: 2% a.m
=====
CONTA : 0294804783
CPF ..: 456
NOME ..: Beltrano
SALDO : R$100
=====
ANIVERSARIO: dia 12
=====
...
```

# Métodos Virtuais



# Métodos Virtuais

```
int main
{
    Conta *agencia[10];
    int i;

    for(i=0; i<10; i++)
    {
        if(i%2==0) agencia[i] = new ContaPoupanca("123", "Fulano", 12, 7, 2016);
        else agencia[i] = new ContaInvestimento("456", "Beltrano", 10);
        agencia[i]->deposito(100);
    }

    for(i=0; i<10; i++)
    {
        agencia[i]->correcao();
    }

    for(i=0; i<10; i++)
    {
        agencia[i]->extrato();
    }
    return 0;
}
```

# Métodos Virtuais

```
int main
{
    Conta *agencia[10];
    int i;

    for(i=0; i<10; i++)
    {
        if(i%2==0) agencia[i] = new ContaPoupanca("123", "Fulano", 12, 7, 2016);
        else agencia[i] = new ContaInvestimento("456", "Beltrano", 10);
        agencia[i]->deposito(100);
    }

    for(i=0; i<10; i++)
    {
        agencia[i]->correcao();
    }

    for(i=0; i<10; i++)
    {
        agencia[i]->extrato();
    }
    return 0;
}
```

**Conta.cpp: In function 'int main()':Conta.cpp:333:21: error: 'class Conta' has no member named 'correcao'      agencia[i]->correcao();**

# Métodos Virtuais Puros

- Métodos virtuais puros são métodos declarados na classe-base, porém que não são implementados na classe-base.
- Você assume o compromisso de implementá-lo na classe derivada.

# Métodos Virtuais Puros

- Métodos virtuais puros são métodos declarados na classe-base, porém que não são implementados na classe-base.
- Você assume o compromisso de implementá-lo na classe derivada.

Conta.cpp: In function 'int main()':

Conta.cpp:341:80: error: cannot allocate an object of abstract type 'ContaInvestimento'

if(i%2==0) agencia[i] = new ContaInvestimento("123", "Fulano", 1, 10, 2);

Conta.cpp:64:7: note: because the following virtual functions are pure within 'ContaInvestimento':

class ContaInvestimento: public Conta

Conta.cpp:29:26: note: virtual void Conta::correcao() virtual void correcao() = 0;

Conta.cpp:342:75: error: cannot allocate an object of abstract type 'ContaPoupanca'

else agencia[i] = new ContaPoupanca("456", "Beltrano", 12, 7, 2016);

Conta.cpp:48:7: note: because the following virtual functions are pure within 'ContaPoupanca':

class ContaPoupanca : public Conta

Conta.cpp:29:26: note: virtual void Conta::correcao() virtual void correcao() = 0;

# Classe Conta

- 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);
        virtual void extrato();
        virtual void correcao() = 0;
        static std::string geraNumero(int);
};

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



# Classe Conta

- 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);
        virtual void extrato();
        virtual void correcao() = 0;
        static std::string geraNumero(int);
};
```

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

← **Método Virtual Puro**

# Classes ContaPoupanca e ContaInvestimento

- Redefinindo um método virtual puro

```
void ContaPoupanca::correcao()
{
    if(!this->isBloqueada())
    {
        this->setSaldo(this->getSaldo()*1.02);
    }
}
```

```
void ContaInvestimento::correcao()
{
    if(!this->isBloqueada())
    {
        this->setSaldo(this->getSaldo()*(1+this->getTxRendimento()/100));
    }
}
```

# Métodos Virtuais Puros

```
int main
{
    Conta *agencia[10];
    int i;

    for(i=0; i<10; i++)
    {
        if(i%2==0) agencia[i] = new ContaPoupanca("123", "Fulano", 12, 7, 2016);
        else agencia[i] = new ContaInvestimento("456", "Beltrano", 10);
        agencia[i]->deposito(100);
    }

    for(i=0; i<10; i++)
    {
        agencia[i]->correcao();
    }

    for(i=0; i<10; i++)
    {
        agencia[i]->extrato();
    }
    return 0;
}
```

# Métodos Virtuais Puros

```
int main
{
    Conta *agencia[10];
    int i;

    for(i=0; i<10; i++)
    {
        if(i%2==0) agencia[i] = new ContaPoupanca("123", "Fulano", 12, 7, 2016);
        else agencia[i] = new ContaInvestimento("456", "Beltrano", 10);
        agencia[i]->deposito(100);
    }

    for(i=0; i<10; i++)
    {
        agencia[i]->correcao();
    }

    for(i=0; i<10; i++)
    {
        agencia[i]->extrato();
    }
    return 0;
}
```

```
=====
CONTA : 8661709620
CPF ..: 123
NOME  .: Fulano
SALDO : R$110
=====
TAXA DE RENDIMENTO: 10%a.m
TAXA DE ADMINISTRACAO: 2%a.m
=====
CONTA : 8661709620
CPF ..: 456
NOME  .: Beltrano
SALDO : R$102
=====
ANIVERSARIO: dia 12
=====
...
```

# Classes Abstratas

- São classes que servem apenas como base para herança de outras classes.
- Não se pode instanciar nenhum objeto de classes abstratas.

# Classes Abstratas

- São classes que servem apenas como base para herança de outras classes.
- Não se pode instanciar nenhum objeto de classes abstratas.
- Em C++, qualquer classe que tenha a definição de pelo menos um método virtual puro é uma classe abstrata.

# Métodos Virtuais

- Em JAVA todos os métodos são virtuais.
- Ou seja, se um método for redefinido em uma subclasse e for invocado através de uma chamada polimórfica, sempre será executado o método da subclasse.

# Classe Conta

```
class Conta
{
    private String numero;
    private String titular;
    private String cpf;
    private boolean bloqueada;
    private float saldo;

    public Conta() { }
    public Conta(String cpf, String nome) { ... }
    public void setNumero(String num) { ... }
    public void setTitular(String nome) { ... }
    public void setCpf(String nCpf) { ... }
    public void setBloqueada(boolean bl) { ... }
    public void setSaldo(float val) { ... }
    public String getNumero() { ... }
    public String getTitular() { ... }
    public String getCpf() { ... }
    public boolean isBloqueada() { ... }
    public float getSaldo() { ... }
    public boolean saque(float valor) { ... }
    public boolean deposito(float valor) { ... }

    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("=====");
    }

    public static String geraNumero(int n) { ... }
}
```



# Classe Derivada ContaPoupanca

```
class ContaPoupanca extends Conta
{
```

```
    private Data aniversario;
```

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

```
    public void setAniversario(int d, int m, int a) { ... }
```

```
    public void setAniversario(Data dt) { ... }
```

```
    public Data getAniversario() { ... }
```

```
    public void extrato()
```

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

```
    public void correcao() { ... }
```

```
}
```

```
class Data
```

```
{
```

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

```
    public Data() { }
```

```
    public Data(int d, int m, int a)
```

```
    public void setDia(int d){ ... }
```

```
    public void setMes(int m){ ... }
```

```
    public void setAno(int a){ ... }
```

```
    public int getDia() { ... }
```

```
    public int getMes() { ... }
```

```
    public int getAno() { ... }
```

```
}
```

# Classe ContaInvestimento

```
class ContaInvestimento extends Conta
{
    private int risco;
    private float txRendimento;
    private float txAdministracao;

    public ContaInvestimento()    {    }

    public ContaInvestimento(String cpf, String tit, int r, float txR, float txA)
    {
        super(cpf, tit);
        this.setRisco(r);
        this.setTxRendimento(txR);
        this.setTxAdministracao(txA);
    }

    public void setRisco(int r) { ... }
    public void setTxRendimento(float txR) { ... }
    public void setTxAdministracao(float txA) { ... }
    public int getRisco() { ... }
    public float getTxRendimento() { ... }
    public float getTxAdministracao() { ... }

    public void extrato()
    {
        super.extrato();
        System.out.println("TAXA DE RENDIMENTO: " + this.getTxRendimento() + "% a.m.");
        System.out.println("TAXA DE ADMINISTRACAO: " + this.getTxAdministracao() + "% a.m.");
        System.out.println("=====");
    }

    public void correcao() { ... }
}
```

# Main

```
public class Main
{
    public static void main(String[] args)
    {
        Conta agencia[] = new Conta[10];
        int i;

        for(i=0; i<10; i++)
        {
            if(i%2==0)
            {
                agencia[i] = new ContaPoupanca("123","Fulano",12,7,2016);
            }
            else
            {
                agencia[i] = new ContaInvestimento("456","Beltrano", 1, 10, 2);
            }
            agencia[i].deposito(100);
        }

        for(i=0; i<10; i++)
        {
            agencia[i].extrato();
        }
    }
}
```

# Main

```
=====
CONTA : 6696538385
CPF ..: 123
NOME ..: Fulano
SALDO : R$100.0
=====
```

```
ANIVERSARIO: dia 12
=====
```

```
CONTA : 7827705612
CPF ..: 456
NOME ..: Beltrano
SALDO : R$100.0
=====
```

```
TAXA DE RENDIMENTO: 10.0% a.m.
TAXA DE ADMINISTRACAO: 2.0% a.m.
=====
```

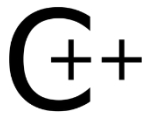
```
...
```

```
=====
CONTA : 1455676025
CPF ..: 123
NOME ..: Fulano
SALDO : R$100.0
=====
```

```
ANIVERSARIO: dia 12
=====
```

```
CONTA : 3748541141
CPF ..: 456
NOME ..: Beltrano
SALDO : R$100.0
=====
```

```
TAXA DE RENDIMENTO: 10.0% a.m.
TAXA DE ADMINISTRACAO: 2.0% a.m.
=====
```

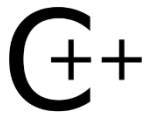


# Main



```
=====
CONTA : 0294804783
CPF ..: 123
NOME .: Fulano
SALDO :R$100
=====
CONTA : 0294804783
CPF ..: 456
NOME .: Beltrano
SALDO : R$100
=====
...
=====
CONTA : 0294804783
CPF ..: 123
NOME .: Fulano
SALDO : R$100
=====
CONTA : 0294804783
CPF ..: 456
NOME .: Beltrano
SALDO : R$100
=====
```

```
=====
CONTA : 6696538385
CPF ..: 123
NOME .: Fulano
SALDO : R$100.0
=====
ANIVERSARIO: dia 12
=====
CONTA : 7827705612
CPF ..: 456
NOME .: Beltrano
SALDO : R$100.0
=====
TAXA DE RENDIMENTO: 10.0% a.m.
TAXA DE ADMINISTRACAO: 2.0% a.m.
=====
...
=====
CONTA : 1455676025
CPF ..: 123
NOME .: Fulano
SALDO : R$100.0
=====
ANIVERSARIO: dia 12
=====
CONTA : 3748541141
CPF ..: 456
NOME .: Beltrano
SALDO : R$100.0
=====
TAXA DE RENDIMENTO: 10.0% a.m.
TAXA DE ADMINISTRACAO: 2.0% a.m.
=====
```



# Main



```
=====
CONTA : 0294804783
CPF ..: 123
NOME  .: Fulano
SALDO :R$100
=====
CONTA : 0294804783
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100
=====
...
=====
CONTA : 0294804783
CPF ..: 123
NOME  .: Fulano
SALDO : R$100
=====
CONTA : 0294804783
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100
=====
```



```
=====
CONTA : 0294804783
CPF ..: 123
NOME  .: Fulano
SALDO :R$100
=====
TAXA DE RENDIMENTO: 10% a.m
TAXA DE ADMINISTRACAO: 2% a.m
=====
CONTA : 0294804783
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100
=====
ANIVERSARIO: dia 12
=====
...
```

Com Método extrato() virtual

```
=====
CONTA : 6696538385
CPF ..: 123
NOME  .: Fulano
SALDO : R$100.0
=====
ANIVERSARIO: dia 12
=====
CONTA : 7827705612
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100.0
=====
TAXA DE RENDIMENTO: 10.0% a.m.
TAXA DE ADMINISTRACAO: 2.0% a.m.
=====
...
=====
CONTA : 1455676025
CPF ..: 123
NOME  .: Fulano
SALDO : R$100.0
=====
ANIVERSARIO: dia 12
=====
CONTA : 3748541141
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100.0
=====
TAXA DE RENDIMENTO: 10.0% a.m.
TAXA DE ADMINISTRACAO: 2.0% a.m.
=====
```



# Main



```
=====
CONTA : 0294804783
CPF ..: 123
NOME  .: Fulano
SALDO :R$100
=====
```

```
=====
CONTA : 0294804783
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100
=====
```

...

```
=====
CONTA : 0294804783
CPF ..: 123
NOME  .: Fulano
SALDO : R$100
=====
```

```
=====
CONTA : 0294804783
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100
=====
```

```
=====
CONTA : 0294804783
CPF ..: 123
NOME  .: Fulano
SALDO :R$100
=====
```

```
TAXA DE RENDIMENTO: 10% a.m
TAXA DE ADMINISTRACAO: 2% a.m
=====
```

```
=====
CONTA : 0294804783
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100
=====
```

```
ANIVERSARIO: dia 12
=====
```

...

Com Método extrato() virtual

**Em JAVA todos os métodos são virtuais.**

```
=====
CONTA : 6696538385
CPF ..: 123
NOME  .: Fulano
SALDO : R$100.0
=====
```

```
ANIVERSARIO: dia 12
=====
```

```
=====
CONTA : 7827705612
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100.0
=====
```

```
TAXA DE RENDIMENTO: 10.0% a.m.
TAXA DE ADMINISTRACAO: 2.0% a.m.
=====
```

...

```
=====
CONTA : 1455676025
CPF ..: 123
NOME  .: Fulano
SALDO : R$100.0
=====
```

```
ANIVERSARIO: dia 12
=====
```

```
=====
CONTA : 3748541141
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100.0
=====
```

```
TAXA DE RENDIMENTO: 10.0% a.m.
TAXA DE ADMINISTRACAO: 2.0% a.m.
=====
```



# Main



```
=====
CONTA : 0294804783
CPF ..: 123
NOME  .: Fulano
SALDO :R$100
=====
CONTA : 0294804783
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100
=====
...
=====
CONTA : 0294804783
CPF ..: 123
NOME  .: Fulano
SALDO : R$100
=====
CONTA : 0294804783
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100
=====
```

```
=====
CONTA : 0294804783
CPF ..: 123
NOME  .: Fulano
SALDO :R$100
=====
TAXA DE RENDIMENTO: 10% a.m
TAXA DE ADMINISTRACAO: 2% a.m
=====
CONTA : 0294804783
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100
=====
ANIVERSARIO: dia 12
=====
...
```

Com Método extrato() virtual

```
=====
CONTA : 6696538385
CPF ..: 123
NOME  .: Fulano
SALDO : R$100.0
=====
ANIVERSARIO: dia 12
=====
CONTA : 7827705612
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100.0
=====
TAXA DE RENDIMENTO: 10.0% a.m.
TAXA DE ADMINISTRACAO: 2.0% a.m.
=====
...
=====
CONTA : 1455676025
CPF ..: 123
NOME  .: Fulano
SALDO : R$100.0
=====
ANIVERSARIO: dia 12
=====
CONTA : 3748541141
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100.0
=====
TAXA DE RENDIMENTO: 10.0% a.m.
TAXA DE ADMINISTRACAO: 2.0% a.m.
=====
```

**Em JAVA todos os métodos são virtuais.**  
**VIRTUAIS ou VIRTUAIS PUROS?**





# Classe Derivada ContaPoupanca

```
class ContaPoupanca extends Conta
{
```

```
    private Data aniversario;
```

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

```
    public void setAniversario(int d, int m, int a) { ... }
```

```
    public void setAniversario(Data dt) { ... }
```

```
    public Data getAniversario() { ... }
```

```
    public void extrato() { ... }
```

```
    public void correcao()
```

```
    {
```

```
        if(!this.isBloqueada())
```

```
        {
```

```
            this.setSaldo(this.getSaldo()*(float)1.02);
```

```
        }
```

```
    }
```

```
}
```

```
class Data
```

```
{
```

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

```
    public Data() { }
```

```
    public Data(int d, int m, int a)
```

```
    public void setDia(int d){ ... }
```

```
    public void setMes(int m){ ... }
```

```
    public void setAno(int a){ ... }
```

```
    public int getDia() { ... }
```

```
    public int getMes() { ... }
```

```
    public int getAno() { ... }
```

```
}
```

# Classe Derivada ContaInvestimento

```
class ContaInvestimento extends Conta
{
    private int risco;
    private float txRendimento;
    private float txAdministracao;

    public ContaInvestimento()    {    }

    public ContaInvestimento(String cpf, String tit, int r, float txR, float txA)
    {
        super(cpf, tit);
        this.setRisco(r);
        this.setTxRendimento(txR);
        this.setTxAdministracao(txA);
    }

    public void setRisco(int r) { ... }
    public void setTxRendimento(float txR) { ... }
    public void setTxAdministracao(float txA) { ... }
    public int getRisco() { ... }
    public float getTxRendimento() { ... }
    public float getTxAdministracao() { ... }
    public void extrato() { ... }

    public void correcao()
    {
        if(!this.isBloqueada())
        {
            this.setSaldo(this.getSaldo()*(1+this.getTxRendimento()/100));
        }
    }
}
```

# Main

```
public class Main
{
    public static void main(String[] args)
    {
        Conta agencia[] = new Conta[10];
        int i;

        for(i=0; i<10; i++)
        {
            if(i%2==0)
            {
                agencia[i] = new ContaPoupanca("123","Fulano",12,7,2016);
            }
            else
            {
                agencia[i] = new ContaInvestimento("456","Beltrano", 1, 10, 2);
            }
            agencia[i].deposito(100);
        }

        for(i=0; i<10; i++)
        {
            agencia[i].correcao();
        }

        for(i=0; i<10; i++)
        {
            agencia[i].extrato();
        }
    }
}
```

# Main

```
public class Main
{
    public static void main(String[] args)
    {
        Conta agencia[] = new Conta[10];
        int i;

        for(i=0; i<10; i++)
        {
            if(i%2==0)
            {
                agencia[i] = new ContaPoupanca("123","Fulano",12,7,2016);
            }
            else
            {
                agencia[i] = new ContaInvestimento("456","Beltrano", 1, 10, 2);
            }
            agencia[i].deposito(100);
        }

        for(i=0; i<10; i++)
        {
            agencia[i].correcao();
        }

        for(i=0; i<10; i++)
        {
            agencia[i].extrato();
        }
    }
}
```

monael:~/workspace/Conta \$ javac Main.java  
Main.java:25: error: cannot find symbol  
agencia[i].correcao();  
                  ^  
symbol:    method correcao()    location: class Conta  
1 error



# Main



```
=====
CONTA : 0294804783
CPF ..: 123
NOME  .: Fulano
SALDO :R$100
=====
```

```
=====
CONTA : 0294804783
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100
=====
```

...

```
=====
CONTA : 0294804783
CPF ..: 123
NOME  .: Fulano
SALDO : R$100
=====
```

```
=====
CONTA : 0294804783
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100
=====
```

```
=====
CONTA : 0294804783
CPF ..: 123
NOME  .: Fulano
SALDO :R$100
=====
```

```
=====
TAXA DE RENDIMENTO: 10% a.m
TAXA DE ADMINISTRACAO: 2% a.m
=====
```

```
=====
CONTA : 0294804783
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100
=====
```

```
=====
ANIVERSARIO: dia 12
=====
```

...

```
=====
CONTA : 6696538385
CPF ..: 123
NOME  .: Fulano
SALDO : R$100.0
=====
```

```
=====
ANIVERSARIO: dia 12
=====
```

```
=====
CONTA : 7827705612
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100.0
=====
```

```
=====
TAXA DE RENDIMENTO: 10.0% a.m.
TAXA DE ADMINISTRACAO: 2.0% a.m.
=====
```

...

```
=====
CONTA : 1455676025
CPF ..: 123
NOME  .: Fulano
SALDO : R$100.0
=====
```

```
=====
ANIVERSARIO: dia 12
=====
```

```
=====
CONTA : 3748541141
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100.0
=====
```

```
=====
TAXA DE RENDIMENTO: 10.0% a.m.
TAXA DE ADMINISTRACAO: 2.0% a.m.
=====
```

Com Método extrato() virtual

**Em JAVA todos os métodos são virtuais.**  
**VIRTUAIS ou VIRTUAIS PUROS?**





# Main



```
=====
CONTA : 0294804783
CPF ..: 123
NOME  .: Fulano
SALDO :R$100
=====
```

```
=====
CONTA : 0294804783
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100
=====
```

...

```
=====
CONTA : 0294804783
CPF ..: 123
NOME  .: Fulano
SALDO : R$100
=====
```

```
=====
CONTA : 0294804783
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100
=====
```

```
=====
CONTA : 0294804783
CPF ..: 123
NOME  .: Fulano
SALDO :R$100
=====
```

```
=====
TAXA DE RENDIMENTO: 10% a.m
TAXA DE ADMINISTRACAO: 2% a.m
=====
```

```
=====
CONTA : 0294804783
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100
=====
```

```
=====
ANIVERSARIO: dia 12
=====
```

...

```
=====
CONTA : 6696538385
CPF ..: 123
NOME  .: Fulano
SALDO : R$100.0
=====
```

```
=====
ANIVERSARIO: dia 12
=====
```

```
=====
CONTA : 7827705612
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100.0
=====
```

```
=====
TAXA DE RENDIMENTO: 10.0% a.m.
TAXA DE ADMINISTRACAO: 2.0% a.m.
=====
```

...

```
=====
CONTA : 1455676025
CPF ..: 123
NOME  .: Fulano
SALDO : R$100.0
=====
```

```
=====
ANIVERSARIO: dia 12
=====
```

```
=====
CONTA : 3748541141
CPF ..: 456
NOME  .: Beltrano
SALDO : R$100.0
=====
```

```
=====
TAXA DE RENDIMENTO: 10.0% a.m.
TAXA DE ADMINISTRACAO: 2.0% a.m.
=====
```

Com Método extrato() virtual

Em JAVA todos os métodos são virtuais.

**VIRTUAIS!** ~~ou VIRTUAIS PUROS?~~



# Métodos Abstratos

- É como se define um método virtual puro na Linguagem JAVA.
- Para declarar um método abstrato em uma classe deve-se usar a palavra reservada `abstract` em sua definição e não definir um bloco de código para ele.

# Métodos Abstratos

- É como se define um método virtual puro na Linguagem JAVA.
- Para declarar um método abstrato em uma classe deve-se usar a palavra reservada `abstract` em sua definição e não definir um bloco de código para ele.

```
public abstract void correcao();
```



# Métodos Abstratos

- É como se define um método virtual puro na Linguagem JAVA.
- Para declarar um método abstrato em uma classe deve-se usar a palavra reservada `abstract` em sua definição e não definir um bloco de código para ele.

```
public abstract void correcao();
```

**Toda classe que possui ao menos um método abstrato deve ser declarada como abstrata.**

# Classe Conta

```
abstract class Conta
{
    private String numero;
    private String titular;
    private String cpf;
    private boolean bloqueada;
    private float saldo;

    public Conta()    {    }
    public Conta(String cpf, String nome)    { ... }
    public void setNumero(String num)    { ... }
    public void setTitular(String nome)    { ... }
    public void setCpf(String nCpf)    { ... }
    public void setBloqueada(boolean bl)    { ... }
    public void setSaldo(float val)    { ... }
    public String getNumero()    { ... }
    public String getTitular()    { ... }
    public String getCpf()    { ... }
    public boolean isBloqueada()    { ... }
    public float getSaldo()    { ... }
    public boolean saque(float valor)    { ... }
    public boolean deposito(float valor)    { ... }
    public void extrato()    { ... }
    public static String geraNumero(int n)    { ... }

    public abstract void correcao();
}
```

# Classe Conta

```
abstract class Conta
```

```
{  
    private String numero;  
    private String titular;  
    private String cpf;  
    private boolean bloqueada;  
    private float saldo;  
  
    public Conta() { }  
    public Conta(String cpf, String nome) { ... }  
    public void setNumero(String num) { ... }  
    public void setTitular(String nome) { ... }  
    public void setCpf(String nCpf) { ... }  
    public void setBloqueada(boolean bl) { ... }  
    public void setSaldo(float val) { ... }  
    public String getNumero() { ... }  
    public String getTitular() { ... }  
    public String getCpf() { ... }  
    public boolean isBloqueada() { ... }  
    public float getSaldo() { ... }  
    public boolean saque(float valor) { ... }  
    public boolean deposito(float valor) { ... }  
    public void extrato() { ... }  
    public static String geraNumero(int n) { ... }  
  
    public abstract void correcao();  
}
```

← Método abstrato (virtual puro)

# Main

```
public class Main
{
    public static void main(String[] args)
    {
        Conta agencia[] = new Conta[10];
        int i;

        for(i=0; i<10; i++)
        {
            if(i%2==0)
            {
                agencia[i] = new ContaPoupanca("123","Fulano",12,7,2016);
            }
            else
            {
                agencia[i] = new ContaInvestimento("456","Beltrano", 1, 10, 2);
            }
            agencia[i].deposito(100);
        }

        for(i=0; i<10; i++)
        {
            agencia[i].correcao();
        }

        for(i=0; i<10; i++)
        {
            agencia[i].extrato();
        }
    }
}
```

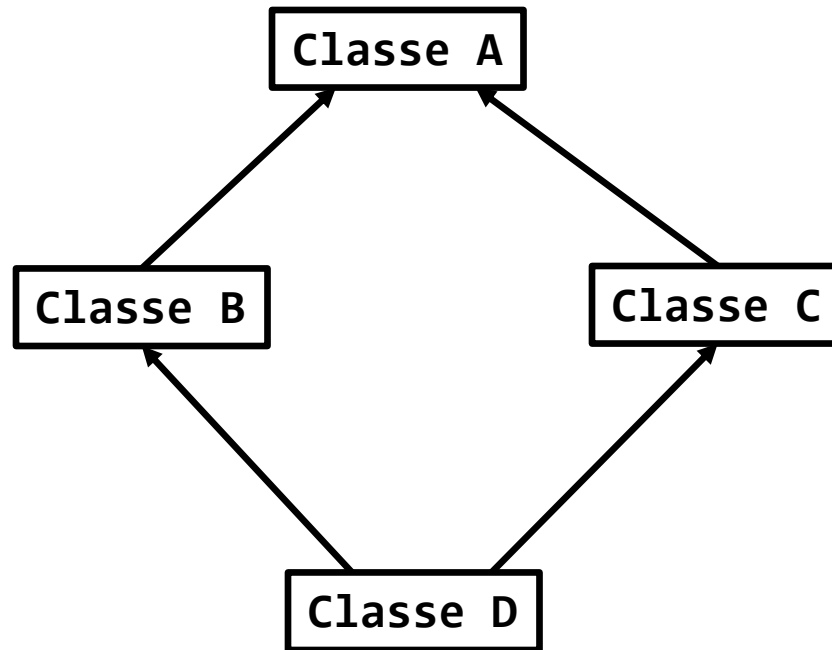
```
=====
CONTA : 0432173871
CPF ..: 123
NOME ..: Fulano
SALDO : R$102.0
=====
ANIVERSARIO: dia 12
=====
CONTA : 1642628592
CPF ..: 456
NOME ..: Beltrano
SALDO : R$110.0
=====
TAXA DE RENDIMENTO: 10.0% a.m.
TAXA DE ADMINISTRACAO: 2.0% a.m.
=====
...
```

# Classes-Base Virtuais

- São necessárias quando uma herança múltipla herda multiplas vezes de uma mesma classe-base.

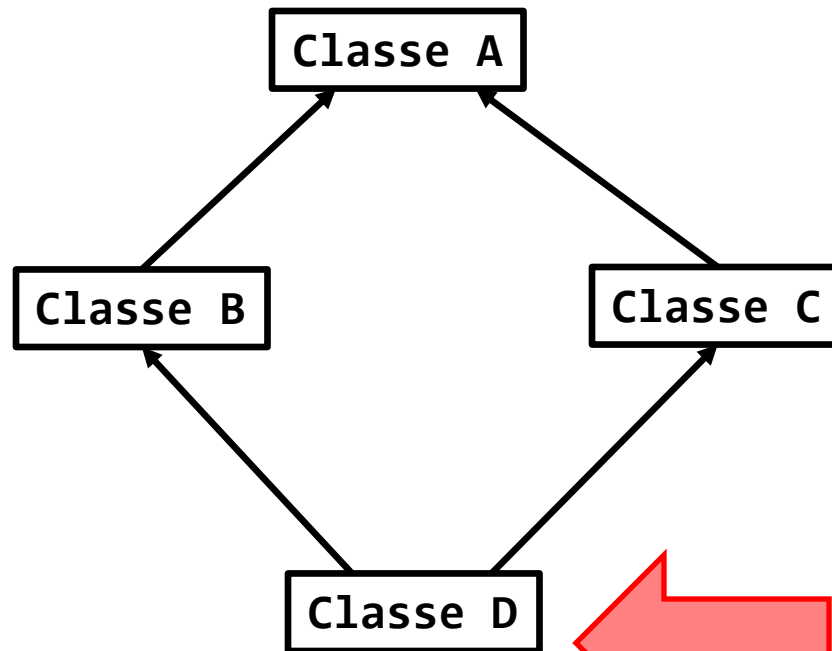
# Classes-Base Virtuais

- São necessárias quando uma herança múltipla herda multiplas vezes de uma mesma classe-base.



# Classes-Base Virtuais

- São necessárias quando uma herança múltipla herda multiplas vezes de uma mesma classe-base.



**Os membros da Classe A  
são duplicados na Classe D**

# Classes-Base Virtuais

```
class ClasseA
{
    public:
        int a, b, c;
        ClasseA() {
            std::cout<<"Classe A"<<std::endl;
        }
};
```

```
class ClasseB : public ClasseA
{
    public:
        int d, e, f;
        ClasseB() {
            std::cout<<"Classe B"<<std::endl;
        }
};
```

```
class ClasseC : public ClasseA
{
    public:
        int g, h, i;
        ClasseC() {
            std::cout<<"Classe C"<<std::endl;
        }
};
```

```
int main()
{
    ClasseD objD;
    objD.a = 10;
    return 0;
}
```

```
class ClasseD : public ClasseB, public ClasseC
{
    public:
        int j, k, l;
        ClasseD() {
            std::cout<<"Classe D"<<std::endl;
        }
};
```



# Classes-Base Virtuais

```
class ClasseA
{
    public:
        int a, b, c;
        ClasseA() {
            std::cout<<"Classe A"<<std::endl;
        }
};
```

```
class ClasseB : public ClasseA
{
    public:
        int d, e, f;
        ClasseB() {
            std::cout<<"Classe B"<<std::endl;
        }
};
```

```
class ClasseC : public ClasseA
{
    public:
        int g, h, i;
        ClasseC() {
            std::cout<<"Classe C"<<std::endl;
        }
};
```

```
int main()
{
    ClasseD objD;
    objD.a = 10;
    return 0;
}
```

```
BaseVirtual.cpp: In function 'int main()':
BaseVirtual.cpp:34:10: error: request for member
'a' is ambiguous      objD.a = 10;
                        ^
BaseVirtual.cpp:6:17: note: candidates are: int
ClasseA::a          int a, b, c;
                        ^
BaseVirtual.cpp:6:17: note:                  int
ClasseA::a
```

```
class ClasseD : public ClasseB, public ClasseC
{
    public:
        int j, k, l;
        ClasseD() {
            std::cout<<"Classe D"<<std::endl;
        }
};
```

# Classes-Base Virtuais

```
class ClasseA
{
    public:
        int a, b, c;
        ClasseA() {
            std::cout<<"Classe A"<<std::endl;
        }
};
```

```
class ClasseB : virtual public ClasseA
{
    public:
        int d, e, f;
        ClasseB() {
            std::cout<<"Classe B"<<std::endl;
        }
};
```

```
class ClasseC : virtual public ClasseA
{
    public:
        int g, h, i;
        ClasseC() {
            std::cout<<"Classe C"<<std::endl;
        }
};
```

```
int main()
{
    ClasseD objD;
    objD.a = 10;
    return 0;
}
```

```
class ClasseD : public ClasseB, public ClasseC
{
    public:
        int j, k, l;
        ClasseD() {
            std::cout<<"Classe D"<<std::endl;
        }
};
```

# Classes-Base Virtuais

```
class ClasseA
{
    public:
        int a, b, c;
        ClasseA() {
            std::cout<<"Classe A"<<std::endl;
        }
};
```

```
class ClasseB : virtual public ClasseA
{
    public:
        int d, e, f;
        ClasseB() {
            std::cout<<"Classe B"<<std::endl;
        }
};
```

```
class ClasseC : virtual public ClasseA
{
    public:
        int g, h, i;
        ClasseC() {
            std::cout<<"Classe C"<<std::endl;
        }
};
```

```
int main()
{
    ClasseD objD;
    objD.a = 10;
    return 0;
}
```

```
Classe A
Classe B
Classe C
Classe D
```

```
class ClasseD : public ClasseB, public ClasseC
{
    public:
        int j, k, l;
        ClasseD() {
            std::cout<<"Classe D"<<std::endl;
        }
};
```

# Funções Amigas

- O conceito de encapsulamento é violado por um mecanismo que permite que funções não-membro de uma classe tenha permissão especial para acessar os atributos de outra classe.
- Ao declarar uma função como amiga, ela terá os mesmos privilégios que funções-membro da classe.
- Para declarar uma função amiga, usa-se a palavra reservada `friend` na definição da classe, informando a assinatura da função amiga.

# Funções Amigas

- Em C++

```
class Tempo
{
    private:
        int segundos;
    public:
        Tempo();
        Tempo(int, int, int);
        void setSegundos(int);
        int getSegundos();
        friend int qtdHoras(Tempo*);
};
```

```
int qtdHoras(Tempo *t)
{
    int i;
    for(i=0; t->segundos>=3600; i++, t->segundos -= 3600);
    return i;
}
```

```
int main()
{
    Tempo *t;
    int h, m, s;
    std::cin >> h >> m >> s;
    t = new Tempo(h, m, s);
    std::cout << qtdHoras(t) << std::endl;
    std::cout << t->getSegundos() << std::endl;
    return 0;
}

Tempo::Tempo(int h, int m, int s)
{
    this->setSegundos(h*3600+m*60+s);
}
```

# Funções Amigas

- Em C++

```
class Tempo
{
    private:
        int segundos;
    public:
        Tempo();
        Tempo(int, int, int);
        void setSegundos(int);
        int getSegundos();
        friend int qtdHoras(Tempo*);
};
```

```
int qtdHoras(Tempo *t)
{
    int i;
    for(i=0; t->segundos>=3600; i++, t->segundos -= 3600);
    return i;
}
```

```
int main()
{
    Tempo *t;
    int h, m, s;
    std::cin >> h >> m >> s;
    t = new Tempo(h, m, s);
    std::cout << qtdHoras(t) << std::endl;
    std::cout << t->getSegundos() << std::endl;
    return 0;
}

Tempo::Tempo(int h, int m, int s)
{
    this->setSegundos(h*3600+m*60+s);
}
```

```
5 59 59
5
3599
```

# Funções Amigas

- Uma das utilidades de uma função amiga é fazer a interface entre classes.
- Uma função não-membro de nenhuma das classes envolvidas que opera sobre os atributos desses classes.

# Funções Amigas

- Funções amigas e sobrecarga de operadores



# Funções Amigas

- Funções amigas e sobrecarga de operadores

```
class Ponto
{
    private:
        float x, y;
    public:
        Ponto();
        Ponto(float, float);
        void setX(float);
        void setY(float);
        float getX();
        float getY();
        Ponto operator +(Ponto);
        Ponto operator +(int);
};
```

```
Ponto Ponto::operator +(int v)
{
    return Ponto(this->getX()+v, this->getY()+v);
}
```

```
Ponto Ponto::operator +(Ponto p)
{
    return Ponto(this->getX()+p.getX(), this->getY()+p.getY());
}
```

# Funções Amigas

- Funções amigas e sobrecarga de operadores

```
int main()
{
    Ponto p1(5,1), p2(2,3), pr;
    pr = p1 + p2;
    std::cout << pr.getX() << " " << pr.getY() << std::endl;
    pr = p1 + 5;
    std::cout << pr.getX() << " " << pr.getY() << std::endl;
    pr = 5 + p1;
    std::cout << pr.getX() << " " << pr.getY() << std::endl;
    return 0;
}
```

# Funções Amigas

- Funções amigas e sobrecarga de operadores

```
int main()
{
    Ponto p1(5,1), p2(2,3), pr;
    pr = p1 + p2;
    std::cout << pr.getX() << " " << pr.getY() << std::endl;
    pr = p1 + 5;
    std::cout << pr.getX() << " " << pr.getY() << std::endl;
    pr = 5 + p1;
    std::cout << pr.getX() << " " << pr.getY() << std::endl;
    return 0;
}
```

7 4  
10 6

# Funções Amigas

- Funções amigas e sobrecarga de operadores

```
int main()
{
    Ponto p1(5,1), p2(2,3), pr;
    pr = p1 + p2;
    std::cout << pr.getX() << " " << pr.getY() << std::endl;
    pr = p1 + 5;
    std::cout << pr.getX() << " " << pr.getY() << std::endl;
    pr = 5 + p1;
    std::cout << pr.getX() << " " << pr.getY() << std::endl;
    return 0;
}
```

```
Ponto.cpp: In function 'int main()':
Ponto.cpp:68:12: error: no match for 'operator+' (operand
types are 'int' and 'Ponto')
    pr = 5 + p1;
           ^
```

# Funções Amigas

- Funções amigas e sobrecarga de operadores

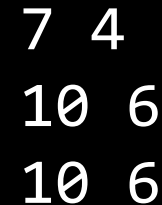
```
class Ponto
{
    private:
        float x, y;
    public:
        Ponto();
        Ponto(float, float);
        void setX(float);
        void setY(float);
        float getX();
        float getY();
        Ponto operator +(Ponto);
        Ponto operator +(int);
        friend Ponto operator +(int, Ponto);
};
```

```
Ponto Ponto::operator +(int v, Ponto p)
{
    return Ponto(p.getX()+v, p.getY()+v);
}
```

# Funções Amigas

- Funções amigas e sobrecarga de operadores

```
class Ponto
{
    private:
        float x, y;
    public:
        Ponto();
        Ponto(float, float);
        void setX(float);
        void setY(float);
        float getX();
        float getY();
        Ponto operator +(Ponto);
        Ponto operator +(int);
        friend Ponto operator +(int, Ponto);
};
```



7 4  
10 6  
10 6

```
Ponto Ponto::operator +(int v, Ponto p)
{
    return Ponto(p.getX()+v, p.getY()+v);
}
```

# Funções Amigas

- Em JAVA

# Funções Amigas

- Em JAVA

**Não há o conceito de Funções Amigas e nem de Classes Amigas**



# Funções Amigas

- Em JAVA

**Não há o conceito de Funções Amigas e nem de Classes Amigas**



# Funções Amigas

- Em JAVA

**Não há o conceito de Funções Amigas e nem de Classes Amigas**



# Funções Amigas

- Em JAVA

**Não há o conceito de Funções Amigas e nem de Classes Amigas**



# Funções Amigas

- Em JAVA



**Não há o conceito de Funções Amigas e nem de Classes Amigas**

