



Universidade Federal do ABC
Centro de Matemática, Computação e Cognição

Programação Orientada a Objetos

Monael Pinheiro Ribeiro, D.Sc.

Sobrecarga

- É a capacidade de POO permitir que exista diversos métodos com o mesmo identificador (nome).
- Em POO o que define um método é a tripla: Identificador do Método, Tipo de Retorno do Método e Lista de Argumentos.
- Deste modo, fica a critério do Compilador eleger quem será o método invocado dada essas três características no momento da chamada.

Sobrecarga

- É a capacidade de POO permitir que exista diversos métodos com o mesmo identificador (nome).
- Em POO o que define um método é a tripla: Identificador do Método, Tipo de Retorno do Método e Lista de Argumentos.
- Deste modo, fica a critério do Compilador eleger quem será o método invocado dada essas três características no momento da chamada.
- A Sobrecarga é um tipo de **Polimorfismo**.

Sobrecarga

- É a capacidade de POO permitir que exista diversos métodos com o mesmo identificador (nome).
- Em POO o que define um método é a tripla: Identificador do Método, Tipo de Retorno do Método e Lista de Argumentos.
- Deste modo, fica a critério do Compilador eleger quem será o método invocado dada essas três características no momento da chamada.
- A Sobrecarga é um tipo de **Polimorfismo**.
 - (poli = muitas e morphos = formas)
 - É quando uma mesma instrução de programa pode assumir vários resultados diferentes.
 - Uma mesma instrução pode chamar métodos diferentes.
 - Um mesmo operador realizar operações sobre tipos de dados diferentes

Sobrecarga

- É a capacidade de POO permitir que exista diversos métodos com o mesmo identificador (nome).
- Em POO o que define um método é a tripla: Identificador do Método, Tipo de Retorno do Método e Lista de Argumentos.
- Deste modo, fica a critério do Compilador eleger quem será o método invocado dada essas três características no momento da chamada.
- Vantagens:
 - Promove o polimorfismo paramétrico;
 - Permite que métodos que realizam tarefas semanticamente iguais tenham o mesmo nome.

Sobrecarga

- É a capacidade de POO permitir que exista diversos métodos com o mesmo identificador (nome).
- Em POO o que define um método é a tripla: Identificador do Método, Tipo de Retorno do Método e Lista de Argumentos.
- Deste modo, fica a critério do Compilador eleger quem será o método invocado dada essas três características no momento da chamada.
- Tipos:
 - Sobrecarga de Métodos Construtores;
 - Sobrecarga de Métodos;
 - Sobrecarga de Operadores

Sobrecarga

- É a capacidade de POO permitir que exista diversos métodos com o mesmo identificador (nome).
- Em POO o que define um método é o trio: Identificador do Método, Tipo de Retorno do Método e Lista de Argumentos.
- Deste modo, fica a critério do Compilador eleger quem será o método invocado dada essas três características no momento da chamada.
- Tipos:
 - **Sobrecarga de Métodos Construtores;**
 - Sobrecarga de Métodos;
 - Sobrecarga de Operadores

Sobrecarga de Construtores

- 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();
        int getMes();
        int getAno();
        void printData();
};
```


Sobrecarga de Construtores

- 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();
        int getMes();
        int getAno();
        void printData();
};
```

```
Data::Data()
{
    int x;
    std::cout << "Informe o dia: ";
    std::cin >> x;
    this->setDia(dia);
    std::cout << "Informe o mes: ";
    std::cin >> x;
    this->setMes(mes);
    std::cout << "Informe o ano: ";
    std::cin >> x;
    this->setAno(ano);
}

Data::Data(int dia, int mes, int ano)
{
    this->setDia(dia);
    this->setMes(mes);
    this->setAno(ano);
}
```

Sobrecarga de Construtores

- 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();
        int getMes();
        int getAno();
        void printData();
};

int main() {
    Data repub(15, 11, 1889), natal, descob(22, 4, 1500);
    repub.printData();
    descob.printData();
    natal.printData();
    return 0;
}
```

```
Data::Data()
{
    int x;
    std::cout << "Informe o dia: ";
    std::cin >> x;
    this->setDia(dia);
    std::cout << "Informe o mes: ";
    std::cin >> x;
    this->setMes(mes);
    std::cout << "Informe o ano: ";
    std::cin >> x;
    this->setAno(ano);
}

Data::Data(int dia, int mes, int ano)
{
    this->setDia(dia);
    this->setMes(mes);
    this->setAno(ano);
}
```



Sobrecarga de Construtores

- Exemplo em JAVA

```
public class Data
{
    private int dia, mes, ano;

    public Data()
    {
        Scanner scan = new Scanner(System.in);
        System.out.print("Informe o dia: ");
        this.setDia(scan.nextInt());
        System.out.print("Informe o mes: ");
        this.setMes(scan.nextInt());
        System.out.print("Informe o ano: ");
        this.setAno(scan.nextInt());
    }

    public Data(int dia, int mes, int ano)
    {
        this.setDia(dia);
        this.setMes(mes);
        this.setAno(ano);
    }
}
```

Sobrecarga de Construtores

- Exemplo em JAVA

```
public class Data
{
    private int dia, mes, ano;

    public Data()
    {
        Scanner scan = new Scanner(System.in);
        System.out.print("Informe o dia: ");
        this.setDia(scan.nextInt());
        System.out.print("Informe o mes: ");
        this.setMes(scan.nextInt());
        System.out.print("Informe o ano: ");
        this.setAno(scan.nextInt());
    }

    public Data(int dia, int mes, int ano)
    {
        this.setDia(dia);
        this.setMes(mes);
        this.setAno(ano);
    }
}
```

```
public static void main(String[] args)
{
    Data repub = new Data(15,11,1889);
    Data natal = new Data();
    Data descob = new Data(22,4, 1500);

    repub.printData();
    descob.printData();
    natal.printData();
}
```



Sobrecarga

- É a capacidade de POO permitir que exista diversos métodos com o mesmo identificador (nome).
- Em POO o que define um método é o trio: Identificador do Método, Tipo de Retorno do Método e Lista de Argumentos.
- Deste modo, fica a critério do Compilador eleger quem será o método invocado dada essas três características no momento da chamada.
- Tipos:
 - Sobrecarga de Métodos Construtores;
 - **Sobrecarga de Métodos;**
 - Sobrecarga de Operadores

Sobrecarga de Métodos

- 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();
        int getMes();
        int getAno();
        void printData();
        void printData(std::string);
        void printData(std::string, char);
};
```

Sobrecarga de Métodos

- Exemplo em C++

```
void Data::printData(std::string formato, char separador)
{
    if(formato.substr(0,2).compare("AA") == 0 &&
        formato.substr(2,2).compare("MM") == 0 &&
        formato.substr(4).compare("DD") == 0)
    {
        std::cout << this->getAno() << separador << this->getMes() << separador << this->getDia() << std::endl;
    }
    else if(formato.substr(0,2).compare("AA") == 0 &&
        formato.substr(2,2).compare("DD") == 0 &&
        formato.substr(4).compare("MM") == 0)
    {
        std::cout << this->getAno() << separador << this->getDia() << separador << this->getMes() << std::endl;
    }
    else if(formato.substr(0,2).compare("DD") == 0 &&
        formato.substr(2,2).compare("MM") == 0 &&
        formato.substr(4).compare("AA") == 0)
    {
        std::cout << this->getDia() << separador << this->getMes() << separador << this->getAno() << std::endl;
    }
}
```

Sobrecarga de Métodos

- Exemplo em C++

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

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


Sobrecarga de Métodos

- Exemplo em C++

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

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

int main()
{
    Data repub(15, 11, 1889), natal, descob(22, 4, 1500);

    repub.printData();
    descob.printData("AAMMDD");
    natal.printData("AADDMM", '-');

    return 0;
}
```

Sobrecarga de Métodos

- Exemplo em C++

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

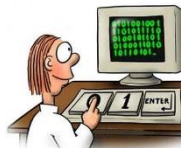
void Data::printData(std::string formato)
{
    this->printData(formato, '/');
}

int main()
{
    Data repub(15, 11, 1889), natal, descob(22, 4, 1500);

    repub.printData();
    descob.printData("AAMMDD");
    natal.printData("AADDMM", '-');

    return 0;
}
```

```
Informe o dia: 25
Informe o mes: 12
Informe o ano: 2016
15/11/1889
1500/4/22
2016-25-12
```



Sobrecarga de Métodos

- Exemplo em JAVA

```
public class Data
{
    private int dia, mes, ano;

    /* ... Construtores e Métodos Sets e Gets ... */

    public void printData(String formato, char separador)
    {
        if(formato.substring(0,2).compareTo("DD") == 0 &&
            formato.substring(2,4).compareTo("MM") == 0 &&
            formato.substring(4).compareTo("AA") == 0)
        {
            System.out.println(this.getDia() + "" + separador + this.getMes() + separador + "" + this.getAno());
        }
        else if(formato.substring(0,2).compareTo("AA") == 0 &&
            formato.substring(2,4).compareTo("MM") == 0 &&
            formato.substring(4).compareTo("DD") == 0)
        {
            System.out.println(this.getAno() + "" + separador + this.getMes() + separador + "" + this.getDia());
        }
        else if(formato.substring(0,2).compareTo("AA") == 0 &&
            formato.substring(2,4).compareTo("DD") == 0 &&
            formato.substring(4).compareTo("MM") == 0)
        {
            System.out.println(this.getAno() + "" + separador + this.getDia() + separador + "" + this.getMes());
        }
    }
}
```

Sobrecarga de Métodos

- Exemplo em JAVA

```
public void printData()  
{  
    this.printData("DDMMAA", '/');  
}
```

```
public void printData(String formato)  
{  
    this.printData(formato, '/');  
}
```

Sobrecarga de Métodos

- Exemplo em JAVA

```
public void printData()  
{  
    this.printData("DDMMAA", '/');  
}  
  
public void printData(String formato)  
{  
    this.printData(formato, '/');  
}  
  
public static void main(String[] args)  
{  
    Data repub = new Data(15,11,1889);  
    Data natal = new Data();  
    Data descob = new Data(22, 4, 1500);  
    repub.printData();  
    descob.printData("AAMMDD");  
    natal.printData("AADDMM", '-');  
}  
}
```

Sobrecarga de Métodos

- Exemplo em JAVA

```
public void printData()  
{  
    this.printData("DDMMAA", '/');  
}  
  
public void printData(String formato)  
{  
    this.printData(formato, '/');  
}  
  
public static void main(String[] args)  
{  
    Data repub = new Data(15,11,1889);  
    Data natal = new Data();  
    Data descob = new Data(22, 4, 1500);  
    repub.printData();  
    descob.printData("AAMMDD");  
    natal.printData("AADDMM", '-');  
}
```

```
Informe o dia: 25  
Informe o mes: 12  
Informe o ano: 2016  
15/11/1889  
1500/4/22  
2016-25-12
```



Sobrecarga

- É a capacidade de POO permitir que exista diversos métodos com o mesmo identificador (nome).
- Em POO o que define um método é o trio: Identificador do Método, Tipo de Retorno do Método e Lista de Argumentos.
- Deste modo, fica a critério do Compilador eleger quem será o método invocado dada essas três características no momento da chamada.
- Tipos:
 - Sobrecarga de Métodos Construtores;
 - Sobrecarga de Métodos;
 - **Sobrecarga de Operadores**

Sobrecarga de Operadores

- A Sobrecarga de Operadores é outro tipo de polimorfismo.
 - O que significa a seguinte instrução?

`r = a + b;`

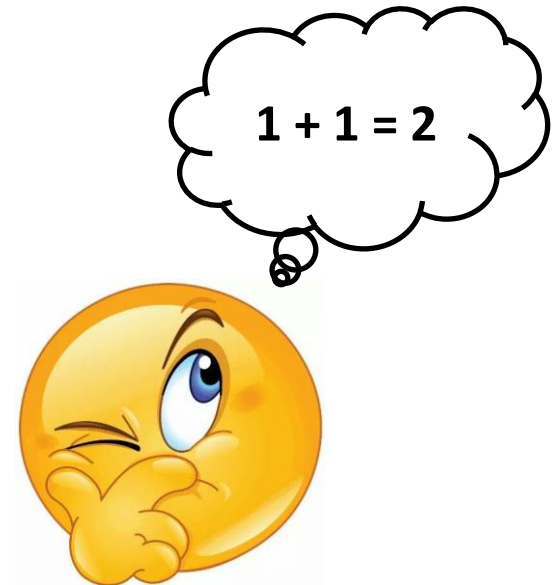
- O que resulta esta expressão?

Sobrecarga de Operadores

- A Sobrecarga de Operadores é outro tipo de polimorfismo.
 - O que significa a seguinte instrução?

`r = a + b;`

- O que resulta esta expressão?



Sobrecarga de Operadores

- A Sobrecarga de Operadores é outro tipo de polimorfismo.
 - O que significa a seguinte instrução?

```
Racional a, b, r;  
r = a + b;
```

- O que resulta esta expressão?

Sobrecarga de Operadores

- A Sobrecarga de Operadores é outro tipo de polimorfismo.
 - O que significa a seguinte instrução?

```
Racional a, b, r;  
r = a + b;
```

- O que resulta esta expressão?

```
Racional.cpp:(.text+0x55a): undefined reference to  
`Racional::operator+(Racional)'collect2: error: ld returned 1  
exit status
```

Sobrecarga de Operadores

```
Racional a, b, r;  
r = a + b;
```



Sobrecarga de Operadores

```
Racional a, b, r;  
r = a + b;  
r++;  
a*=b;
```



Sobrecarga de Operadores

- A Sobrecarga de Operadores é um tipo de polimorfismo.

```
#include <iostream>
int main()
{
    Racional a, b, r;
    int p, q;

    std::cout << "Informe o racional A: " << std::endl;
    std::cin >> p >> q;
    a.setNumerador(p);
    a.setDenominador(q);

    std::cout << "Informe o racional B: " << std::endl;
    std::cin >> p >> q;
    b.setNumerador(p);
    b.setDenominador(q);

    r = a + b;
    std::cout << "R = A + B = " << r.getNumerador() << " " << r.getDenominador() << std::endl;
    r++;
    std::cout << "R = R + 1 = " << r.getNumerador() << " " << r.getDenominador() << std::endl;
    return 0;
}
```

**Isso não seria muito mais
claro, fácil, legível e
intuitivo ...**

Sobrecarga de Operadores

- A Sobrecarga de Operadores é um tipo de polimorfismo.

```
#include <iostream>
int main()
{
    Racional a, b, r;
    int p, q;

    std::cout << "Informe o racional A: " << std::endl;
    std::cin >> p >> q;
    a.setNumerador(p);
    a.setDenominador(q);

    std::cout << "Informe o racional B: " << std::endl;
    std::cin >> p >> q;
    b.setNumerador(p);
    b.setDenominador(q);

    r = a.soma(b);
    std::cout << "R = A + B = " << r.getNumerador() << " " << r.getDenominador() << std::endl;
    r = r.soma(Racional(1,1));
    std::cout << "R = R + 1 = " << r.getNumerador() << " " << r.getDenominador() << std::endl;
    return 0;
}
```

... que isso?

Sobrecarga de Operadores

- Uma deficiência das Linguagens de Programação é não permitir que se use os operadores de uma forma diferente do que eles foram projetados.
- Especialmente quando estamos lidando com um tipo de dado definido pelo usuário (Estruturas ou Classes).
- Sobrecarregar um Operador é justamente redefinir um operador para que ele opere em outros tipos de dados.

Sobrecarga de Operadores

- **Limitações**

- Não é possível mudar a aridade de um operador, ou seja, transformar um operador binário em unário.
- Não é permitido criar novos operadores, ou seja, só é permitido sobrecarregar operadores que já existem na linguagem.
- Não é possível alterar a ordem de precedência original dos operadores, ou seja, a nova definição está submissa às regras de precedências originais.
- Não é permitido sobrecarregar qualquer operador. Os operadores `(.)`, `(::)` e `(?:)` não aceitam sobrecarga.



Sobrecarga de Operadores

- **Sobrecarga de Operadores Binários**

- A sobrecarga de operadores se faz definindo um método usando como identificador do método a palavra reservada **operator** seguido do operador a ser sobrecarregado.

`<tipo de retorno> operator <operador> (<argumentos>);`

- Exemplos

```
Racional operator + (Racional);  
Ponto operator ++();  
Aluno operator = (Aluno);
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores Binários em C++**

Sobrecarga de Operadores

- **Sobrecarga de Operadores Binários em C++**

```
class Racional
{
    private:
        int numerador, denominador;
    public:
        /* metodos construtores*/
        /* metodos set e get*/
        /* metodos especificos*/

        Racional operator + (Racional);
};
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores Binários em C++**

```
class Racional
{
    private:
        int numerador, denominador;
    public:
        /* metodos construtores*/
        /* metodos set e get*/
        /* metodos especificos*/

        Racional operator + (Racional);
};
...
Racional Racional::operator + (Racional q)
{
    Racional r;
    r.setNumerador(this->getNumerador()*q.getDenominador() + this->getDenominador()*q.getNumerador());
    r.setDenominador(this->getDenominador()*q.getDenominador());
    return r;
}
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores Binários em C++**

```
class Racional
{
private:
    int numerador, denominador;
public:
    /* metodos construtores*/
    /* metodos set e get*/
    /* metodos especificos*/

    Racional operator + (Racional);
};
...
Racional Racional::operator + (Racional q)
{
    Racional r;
    r.setNumerador(this->getNumerador()*q.getDenominador() + this->getDenominador()*q.getNumerador());
    r.setDenominador(this->getDenominador()*q.getDenominador());
    return r;
}

int main()
{
    Racional f1(1, 5), f2(2, 10), fr;
    fr = f1 + f2;
    std::cout << fr.getNumerador << " " << fr.getDenominador() << std::endl;
}
```



Sobrecarga de Operadores

- **Sobrecarga de Operadores Binários em C++**

`fr = f1.soma(f2);`

```
int main()
{
    Racional f1(1, 5), f2(2, 10), fr;
    fr = f1 + f2;
    std::cout << fr.getNumerador << " " << fr.getDenominador() << std::endl;
}
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores Binários em C++**

`fr = f1.soma(f2);`

`fr = f1 + f2;`

```
int main()
{
    Racional f1(1, 5), f2(2, 10), fr;
    fr = f1 + f2;
    std::cout << fr.getNumerador << " " << fr.getDenominador() << std::endl;
}
```


Sobrecarga de Operadores

- **Sobrecarga de Operadores Binários em C++**

`fr = f1.soma(f2);`

`fr = f1 + f2;`

`fr = f1.operator+(f2);`

```
int main()
{
    Racional f1(1, 5), f2(2, 10), fr;
    fr = f1 + f2;
    std::cout << fr.getNumerador << " " << fr.getDenominador() << std::endl;
}
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores Binários em JAVA**

Sobrecarga de Operadores

- **Sobrecarga de Operadores Binários em JAVA**

**A linguagem JAVA não permite
a sobrecarga de seus operadores**

Sobrecarga de Operadores

- Sobrecarga de Operadores Binários em JAVA

A linguagem JAVA não permite a sobrecarga de seus operadores



Sobrecarga de Operadores

- **Sobrecarga de Operadores Unários**

- A sobrecarga de operadores unários se faz da mesma forma que a sobrecarga dos operadores binários.
- No entanto, há um pequeno complicador quando sobrecarrega-se os operadores de incremento e decremento unários.
- Esse complicador reside no fato que ambos operadores têm comportamento diferente quando são prefixados ou pós-fixados.
- Além disso, deve-se orientar o compilador qual deles se deseja sobrecarregar.

Você sabia?

- **Operadores de Incremento e Decremento**

```
int main()
{
    int i=0;
    i++;
    std::cout << "i = " << i << std::endl;
    ++i;
    std::cout << "i = " << i << std::endl;
    i--;
    std::cout << "i = " << i << std::endl;
    --i;
    std::cout << "i = " << i << std::endl;
    return 0;
}
```

Você sabia?

- **Operadores de Incremento e Decremento**

```
int main()
{
    int i=0;
    i++;
    std::cout << "i = " << i << std::endl;
    ++i;
    std::cout << "i = " << i << std::endl;
    i--;
    std::cout << "i = " << i << std::endl;
    --i;
    std::cout << "i = " << i << std::endl;
    return 0;
}
```

```
i = 1
i = 2
i = 1
i = 0
```

Você sabia?

- Operadores de Incremento e Decremento

```
int main()
{
    int i=0;
    i++;
    std::cout << "i = " << i << std::endl;
    ++i;
    std::cout << "i = " << i << std::endl;
    i--;
    std::cout << "i = " << i << std::endl;
    --i;
    std::cout << "i = " << i << std::endl;
    return 0;
}
```

```
i = 1
i = 2
i = 1
i = 0
```

São iguais. Ufa!



Você sabia?

- Operadores de Incremento e Decremento

```
int main()
{
    int i=0;
    i++;
    std::cout << "i = " << i << std::endl;
    ++i;
    std::cout << "i = " << i << std::endl;
    i--;
    std::cout << "i = " << i << std::endl;
    --i;
    std::cout << "i = " << i << std::endl;
    return 0;
}
```

```
i = 1
i = 2
i = 1
i = 0
```

~~São iguais. Ufa!~~

**Não! São diferentes.
O retorno os diferencia.**



Você sabia?

- **Operadores de Incremento e Decremento**

```
int main()
{
    int i=0, ret;
    ret = i++;
    std::cout << "i = " << i << "ret = " << ret << std::endl;
    ret = ++i;
    std::cout << "i = " << i << "ret = " << ret << std::endl;
    ret = i--;
    std::cout << "i = " << i << "ret = " << ret << std::endl;
    ret = --i;
    std::cout << "i = " << i << "ret = " << ret << std::endl;
    return 0;
}
```



Você sabia?

- Operadores de Incremento e Decremento

```
int main()
{
    int i=0, ret;
    ret = i++;
    std::cout << "i = " << i << "ret = " << ret << std::endl;
    ret = ++i;
    std::cout << "i = " << i << "ret = " << ret << std::endl;
    ret = i--;
    std::cout << "i = " << i << "ret = " << ret << std::endl;
    ret = --i;
    std::cout << "i = " << i << "ret = " << ret << std::endl;
    return 0;
}
```

```
i = 1 ret = 0
i = 2 ret = 2
i = 1 ret = 2
i = 0 ret = 0
```



Você sabia?

- **Operadores de Incremento e Decremento**

`i++;`



Primeiro retorna o valor de i e depois incrementa o valor de i.

`++i;`



Primeiro incrementa o valor de i e depois retorna o valor de i.

Sobrecarga de Operadores

- **Sobrecarga de Operadores Unários**

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

Sobrecarga de Operadores

- **Sobrecarga de Operadores Unários**

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

Sobrecarga do operador ++ prefixado

Sobrecarga do operador ++ pós-fixado

Sobrecarga de Operadores

- **Sobrecarga de Operadores Unários**

```
Ponto Ponto::operator ++()  
{  
    this->setX(this->getX()+1);  
    this->setY(this->getY()+1);  
    return *this;  
}
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores Unários**

```
Ponto Ponto::operator ++()  
{  
    this->setX(this->getX()+1);  
    this->setY(this->getY()+1);  
    return *this;  
}
```

```
Ponto Ponto::operator ++(int)  
{  
    Ponto r(this->getX(), this->getY());  
    this->setX(this->getX()+1);  
    this->setY(this->getY()+1);  
    return r;  
}
```


Sobrecarga de Operadores

- **Sobrecarga de Operadores Unários**

```
int main()
{
    Ponto p1(1,1), p2(1,1), pr1, pr2;

    pr1 = p1++;

    std::cout << "p1(" << p1.getX() << "," << p1.getY() << ")" << std::endl;
    std::cout << "pr1(" << pr1.getX() << "," << pr1.getY() << ")" << std::endl;

    pr2 = ++p2;

    std::cout << "p2(" << p2.getX() << "," << p2.getY() << ")" << std::endl;
    std::cout << "pr2(" << pr2.getX() << "," << pr2.getY() << ")" << std::endl;

    return 0;
}
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores Unários**

```
int main()
{
    Ponto p1(1,1), p2(1,1), pr1, pr2;

    pr1 = p1++;

    std::cout << "p1(" << p1.getX() << "," << p1.getY() << ")" << std::endl;
    std::cout << "pr1(" << pr1.getX() << "," << pr1.getY() << ")" << std::endl;

    pr2 = ++p2;

    std::cout << "p2(" << p2.getX() << "," << p2.getY() << ")" << std::endl;
    std::cout << "pr2(" << pr2.getX() << "," << pr2.getY() << ")" << std::endl;

    return 0;
}
```



```
p1(2,2)
pr1(1,1)
p2(2,2)
pr2(2,2)
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Conversão)**
 - O operador de atribuição (=) faz atribuições entre tipos iguais:

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Conversão)**
 - O operador de atribuição (=) faz atribuições entre tipos iguais:

```
int i=7, j;  
j = i;
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Conversão)**

- O operador de atribuição (=) faz atribuições entre tipos iguais:

```
int i=7, j;  
j = i;
```

- Ele também pode atribuir objetos de tipos iguais em uma única instrução:

```
Racional f1(5,12), f2;  
f2 = f1;
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Conversão)**

- O operador de atribuição (=) faz atribuições entre tipos iguais:

```
int i=7, j;  
j = i;
```

- Ele também pode atribuir objetos de tipos iguais em uma única instrução:

```
Racional f1(5,12), f2;  
f2 = f1;
```

- E quando a atribuição é feita entre tipos distintos ???

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Conversão)**
 - O que acontece nesta situação?

```
float var = 'A';
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Conversão)**

- O que acontece nesta situação?

```
float var = 'A';
```

- O compilador localiza o código ASCII do caractere 'A', converte em um tipo float e atribui para a variável em questão.
- Isto é chamado de casting implícito ou conversão implícita.

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Conversão)**

- O que acontece nesta situação?

```
float var = 'A';
```

- O compilador localiza o código ASCII do caractere 'A', converte em um tipo float e atribui para a variável em questão.
- Isto é chamado de casting implícito ou conversão implícita.
- Os compiladores contam com diversas rotinas de casting implícitos. Para saber quais, consulte a documentação da Linguagem.

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Conversão)**

- Porém, não são todas as conversões que têm casting implícitos.
- Nestes casos usamos um operador de casting para forçar uma conversão.

```
float var = float('A');
```

ou

```
float var = (float)'A';
```

- Esta conversão é chamada de casting explícito.

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Conversão)**

- Porém, não são todas as conversões que têm casting implícitos.
- Nestes casos usamos um operador de casting para forçar uma conversão.
- Esta conversão é chamada de casting explícito.
- Isto não é novidade ...

```
int a=5, b=2;  
float divReal = a/b;  
std::cout << divReal << std::endl;
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Conversão)**

- Porém, não são todas as conversões que têm casting implícitos.
- Nestes casos usamos um operador de casting para forçar uma conversão.
- Esta conversão é chamada de casting explícito.
- Isto não é novidade ...

```
int a=5, b=2;  
float divReal = a/b;  
std::cout << divReal << std::endl;
```

2.000

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Conversão)**

- Porém, não são todas as conversões que têm casting implícitos.
- Nestes casos usamos um operador de casting para forçar uma conversão.
- Esta conversão é chamada de casting explícito.
- Isto não é novidade ...

```
int a=5, b=2;  
float divReal = a/(float)b;  
std::cout << divReal << std::endl;
```

2.500

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Obj → prim)**
 - Na definição da classe, pode-se instruir o compilador como ele deve se comportar em caso de conversão de Objetos para um tipo primitivo.
 - Isso se realiza sobrecarregando os operadores de casting correspondentes.
 - Por exemplo:

```
Racional fracao(4,3);  
float f;  
double d;  
f = fracao;  
d = fracao;
```
 - Como estou atribuindo um objeto da classe Racional para uma variável do tipo float e double, deve-se sobrecarregar esses operadores de casting para o compilador agir corretamente no casting implícito entre os tipos `double ← Racional` e `float ← Racional`.

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Obj → prim)**

- Exemplo:

```
class Racional
{
    private:
        int numerador, denominador;
    public:
        ...
        operator float();
        operator double();
};
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Obj → prim)**
 - Exemplo:

```
class Racional
{
    private:
        int numerador, denominador;
    public:
        ...
        operator float();
        operator double();
};
```



Perceba que não há tipo de retorno para este operador...

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Obj → prim)**
 - Exemplo:

```
Racional::operator float()
{
    return this->getNumerador()/float(this->getDenominador());
}
```

```
Racional::operator double()
{
    return this->getNumerador()/double(this->getDenominador());
}
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Obj → prim)**
 - Exemplo:

```
int main()
{
    Racional fracao(4,3);
    float f = fracao;
    double d = fracao;
    std::cout << "f = " << f << "\nd = " << d << std::endl;
}
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Obj → prim)**
 - Exemplo:

```
int main()
{
    Racional fracao(4,3);
    float f = fracao;
    double d = fracao;
    std::cout << "f = " << f << "\nd = " << d << std::endl;
}
```

```
f = 1.33333
d = 1.33333
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Obj → prim)**
 - Exemplo:

```
int main()
{
    Racional fracao(4,3);
    float f = fracao;
    double d = fracao;
    std::cout << "f = " << f << "\nd = " << d << std::endl;
}
```


 **Casting Implícito**

```
f = 1.33333
d = 1.33333
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Obj → prim)**
 - Exemplo:

```
int main()
{
    Racional fracao(4,3);
    float f = float(fracao);
    double d = double(fracao);
    std::cout << "f = " << f << "\nd = " << d << std::endl;
}
```



Casting Explícito

```
f = 1.33333
d = 1.33333
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Obj → prim)**
 - Exemplo:

```
int main()
{
    Racional fracao(4,3);
    float f = (float)fracao;
    double d = (double)fracao;
    std::cout << "f = " << f << "\nd = " << d << std::endl;
}
```

← **Casting Explícito**



```
f = 1.33333
d = 1.33333
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (prim → Obj)**
 - Para se converter tipos primitivos em Objetos deve-se sobrecarregar o operador de atribuição (=).

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (prim → Obj)**
 - Para se converter tipos primitivos em Objetos deve-se sobrecarregar o operador de atribuição (=).

```
class Racional
{
    private:
        int numerador, denominador;
    public:
        ...
        void operator =(int);
};
```


Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (prim → Obj)**
 - Para se converter tipos primitivos em Objetos deve-se sobrecarregar o operador de atribuição (=).

```
void Racional::operator =(int v)
{
    this->setNumerador(v);
    this->setDenominador(1);
}
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (prim → Obj)**
 - Para se converter tipos primitivos em Objetos deve-se sobrecarregar o operador de atribuição (=).

```
void Racional::operator =(int v)
{
    this->setNumerador(v);
    this->setDenominador(1);
}
```

```
int main()
{
    int valor = 7;
    Racional fr = valor;
    std::cout << fr.getNumerador() << " " << fr.getDenominador() << std::endl;
}
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (prim → Obj)**
 - Para se converter tipos primitivos em Objetos deve-se sobrecarregar o operador de atribuição (=).

```
void Racional::operator =(int v)
{
    this->setNumerador(v);
    this->setDenominador(1);
}
```

7/1

```
int main()
{
    int valor = 7;
    Racional fr = valor;
    std::cout << fr.getNumerador() << " " << fr.getDenominador() << std::endl;
}
```



Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Obj → Obj)**
 - Também pode-se fazer casting entre objetos de classes definidas pelo usuário. Para isso há duas formas:
 1. Criar um operador de casting com o nome da classe “destino” da conversão dentro da classe “origem”.
 2. Criar um construtor na classe “destino” da conversão que recebe um objeto da classe “origem”.
 - “origem” = objeto da classe que será convertido
 - “destino” = objeto da classe convertida.
 - `objDestino = objOrigem;`
 - Quando se deseja que a conversão seja bidirecional, então implementa-se as duas formas em uma mesma classe.

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Obj → Obj)**

```
class DataJuliana
{
    private:
        int diaJuliano;
    public:
        DataJuliana();
        DataJuliana(int);
        void setDiaJuliano(int);
        int getDiaJuliano();
};
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Obj → Obj)**

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

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Obj → Obj)**

```
class DataGregoriana  
{
```

```
    private:
```

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

```
    public:
```

```
        DataGregoriana();
```

```
        DataGregoriana(int, int, int);
```

```
        DataGregoriana(DataJuliana); ← Construtor Conversor
```

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

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

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

```
        int getDia();
```

```
        int getMes();
```

```
        int getAno();
```

```
        operator DataJuliana(); ← Operador Conversor
```

```
};
```

Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Obj → Obj)**

```
DataGregoriana::DataGregoriana(DataJuliana dt)
{
    /* algoritmo para converter Data Juliana em Gregoriana*/
    /* dtGregorian = dtJulian */
    /*      ^           ^      */
    /*      this       dt (arg) */
}
```


Sobrecarga de Operadores

- **Sobrecarga de Operadores de Casting (Obj → Obj)**

```
DataGregoriana::DataGregoriana(DataJuliana dt)
{
    /* algoritmo para converter Data Juliana em Gregoriana*/
    /* dtGregorian = dtJulian */
    /*      ^           ^      */
    /*      this      dt (arg) */
}
```

```
DataGregoriana::operator DataJuliana()
{
    /* algoritmo para converter Data Gregoriana em Juliana*/
    /* dtJulian = dtGregorian */
    /*      ^           ^      */
    /*will returned      this  */
}
```