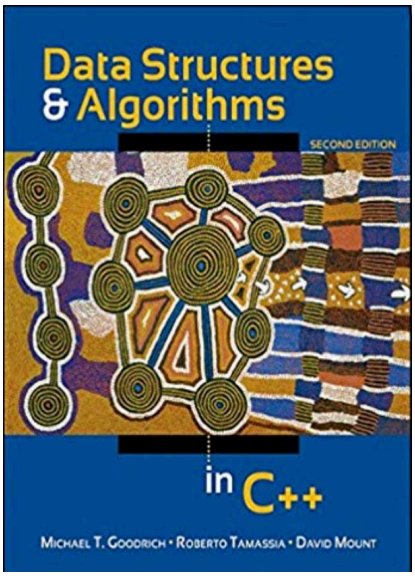
FACULDADE DE COMPUTAÇÃO E INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
ESTRUTURA DE DADOS I – Aula 03 – 2º SEMESTRE/2018
PROF. LUCIANO SILVA/MARCIO PORTO FEITOSA

TEORIA: HERANÇA SIMPLES

Nossos **objetivos** nesta aula são:

- conhecer o conceito de herança simples em C++
- refatorar a implementação dos tipos abstratos de dados Natural e Inteiro com herança simples



Para esta aula, usamos como referência o **Capítulo 1** do nosso livro-texto:

GOODRICH,M., **Data Structures and Algorithms**. 2.ed. New York: Wiley, 2011.

Não deixem de ler este capítulo depois desta aula!

HERANÇA SIMPLES EM C++

- O conceito de **herança** é uma das palavras-chave em Programação Orientada a Objetos (PPO) e, em particular, para a linguagem C++, que pertence ao paradigma orientado a objetos.

- Com o mecanismo de herança, vamos conseguir fazer reúso de implementações, herdando atributos e métodos de classes já implementadas.

- Quando **herdamos atributos e métodos de somente uma classe**, dizemos que a **herança é simples**. Quando a **herança** acontece a **partir de duas ou mais classes**, dizemos que a **herança é múltipla**. A linguagem C++ permite trabalhar tanto com herança simples quanto herança múltipla. Outras linguagens orientadas a objetos (OO) permitem somente herança simples como, por exemplo, a linguagem Java.

- Vamos retomar as implementações dos tipos abstratos realizados nas aulas anteriores, já com modularização e sobrecarga de operadores:

natural.h

```
#ifndef NATURAL_H
#define NATURAL_H

class Natural{

   private:
      unsigned int valor;


   public:
      Natural(int v);
      ~Natural();
      unsigned int getValor();
      Natural suc();
      Natural operator+(Natural n);
};

#endif
```

natural.cpp

```
#include "natural.h"

Natural::Natural(int v){
   valor=v;
}

Natural::~Natural(){}

unsigned int Natural::getValor(){
      return valor;
}

Natural Natural::suc(){
   Natural n(valor+1);
   return n;
}

Natural Natural::operator+(Natural n){
   Natural s(valor+n.getValor());
   return s;
}
```

inteiro.h

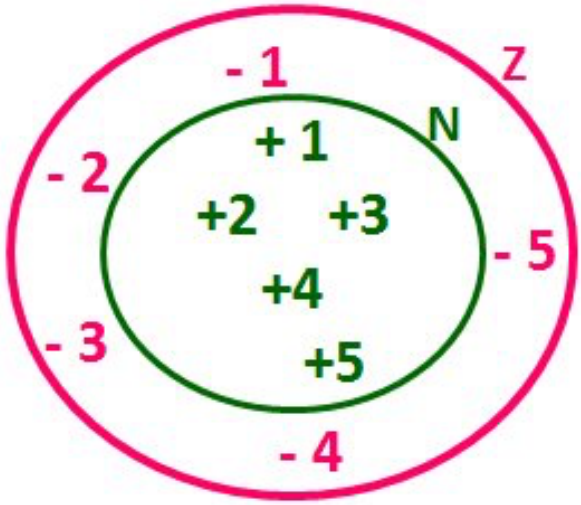
```
#ifndef INTEIRO_H
#define INTEIRO_H

class Inteiro{

   private:
      int valor;


   public:
      Inteiro(int v);
      ~Inteiro();
      int getValor();
      Inteiro suc();
      Inteiro pred();
      Inteiro operator+(Inteiro n);
      Inteiro operator-(Inteiro n);
};
#endif
```

inteiro.cpp

```
#include "inteiro.h"

Inteiro:: Inteiro (int v){
   valor=v;
}

Inteiro:: Inteiro (int v){
   valor=v;
}

Inteiro::~ Inteiro (){}

int Inteiro::getValor(){
      return valor;
}

Inteiro Inteiro::suc(){
   Inteiro n(valor+1);
   return n;
}

Inteiro Inteiro::pred(){
```

	Inteiro n(valor-1);
	return n;
	}
	
	Inteiro Inteiro::operator+(Inteiro n){
	Inteiro s(valor+n.getValor());
	return s;
	}
	
	Inteiro Inteiro::operator-(Inteiro n){
	Inteiro s(valor-n.getValor());
	return s;
	}

- Sabemos, da disciplina de Matemática Discreta e da Teoria dos Números, que existe uma **inclusão** entre o conjunto dos números inteiros e naturais.



- Se observarmos atentamente esta inclusão, podemos concluir que:
  - todo número natural também é um número inteiro, pois todos os números naturais são iguais aos números inteiros positivos
  - existem números inteiros (negativos) que não números naturais

- Assim, podemos modelar estes dois tipos abstratos de dados da mesma forma, colocando um novo **qualificador** chamado **sinal**. Se o sinal for positivo, podemos ter tanto um número inteiro positivo quanto um natural. Se o sinal for negativo, teremos somente números inteiros negativos.

- Vamos, então, refatorar a classe Inteiro para o seguinte código:

inteiro.h

```
#ifndef INTEIRO_H
#define INTEIRO_H

class Inteiro{

   private:
      int valor;
      char sinal;

   public:
      Inteiro(unsigned int v,char sinal);
      ~Inteiro();
      int getValor();
      Inteiro suc();
      Inteiro pred();
      Inteiro operator+(Inteiro n);
      Inteiro operator-(Inteiro n);
};
#endif
```

inteiro.cpp

```
#include "inteiro.h"
#include <stdlib.h>

Inteiro:: Inteiro (unsigned int v,char sinal){
   if (sinal=='+')
      valor = v;
   else if (sinal=='-')
         valor=-v;
}

Inteiro::~ Inteiro (){}

int Inteiro::getValor(){
     return valor;
}

Inteiro Inteiro::suc(){
   int v= valor+1;
   Inteiro n(abs(v), v<0?'-':'+');
   return n;
}

Inteiro Inteiro::pred(){
   int v= valor-1;
   Inteiro n(abs(v), v<0?'-':'+');
   return n;
}

Inteiro Inteiro::operator+(Inteiro n){
   int v= valor+n.getValor();
   Inteiro s(abs(v), v<0?'-':'+');
   return s;
}

Inteiro Inteiro::operator-(Inteiro n){
   Int v = valor-n.getValor();
   Inteiro s(abs(v), v<0?'-':'+');
   return s;
}
```

- Para instanciar um número positivo, fazemos: Inteiro num(3,'+'); . Para um negativo, fazemos Inteiro num(3,'-');

- Vamos, agora, refatorar a classe Natural para usar o mecanismo de herança (também chamado de polimorfismo por inclusão).

natural.h	**natural.cpp**

#ifndef NATURAL_H
#define NATURAL_H
#include "inteiro.h"
 	
class **Natural: public Inteiro**{
 	
public:
Natural(int v);
~Natural();
 	
};
 	
#endif
	#include "natural.h"
	
	Natural::Natural(int v):**Inteiro(v,'+'**){
	}
	
	Natural::~Natural(){}

EXERCÍCIO COM DISCUSSÃO EM DUPLAS

Escreva um módulo principal main.cpp para testar a implementação da herança.

EXERCÍCIO COM DISCUSSÃO EM DUPLAS

Considere a seguinte implementação do tipo abstrato de dado Racional, que abstrai o conceito de números fracionários a/b:

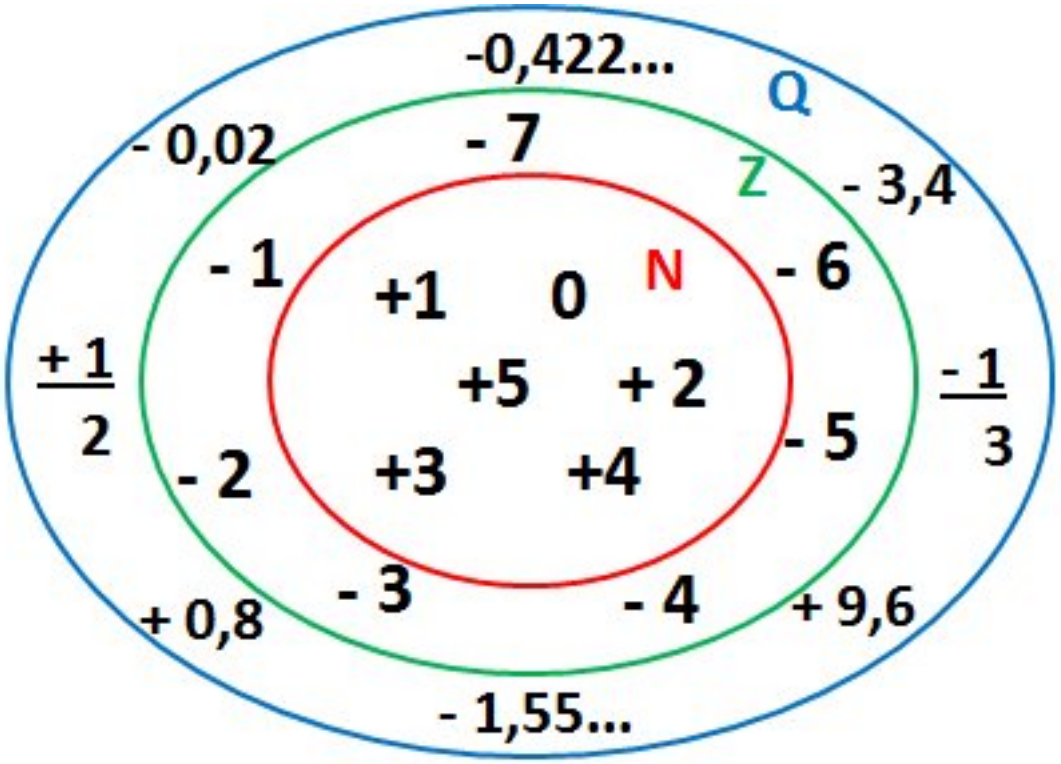
racional.h

```
#ifndef RACIONAL_H
#define RACIONAL_H

class Racional{

   private:
      int a,b;
      char sinal;

   public:
      Racional(int va,int vb,,char sinal);
      ~Racional();
      int getA();
      int getB();
      Racional operator+(Racional n);
      Racional operator-(Racional n);
};
#endif
```

racional.cpp

```
#include "inteiro.h"
#include <stdlib.h>

Racional::Racional(int va,int vb,char sinal){
  a=va;
  b=vb;
  if (sinal=='-')
     a=-a;
}

Racional::~ Racional (){}

int Racional::getA(){
     return a;
}

int Racional::getB(){
     return b;
}

Racional Racional::operator+(Racional n){
   int va= a*n.getB()+b*n.getA();
   int vb=b*n.getB();
   Racional s(abs(va),abs(vb), (va/vb)<0?'-':'+');
   return s;
}

Racional Racional::operator-(Racional n){
   int va= a*n.getB()-b*n.getA();
   int vb=b*n.getB();
   Racional s(abs(va),abs(vb), (va/vb)<0?'-':'+');
   return s;
}
```

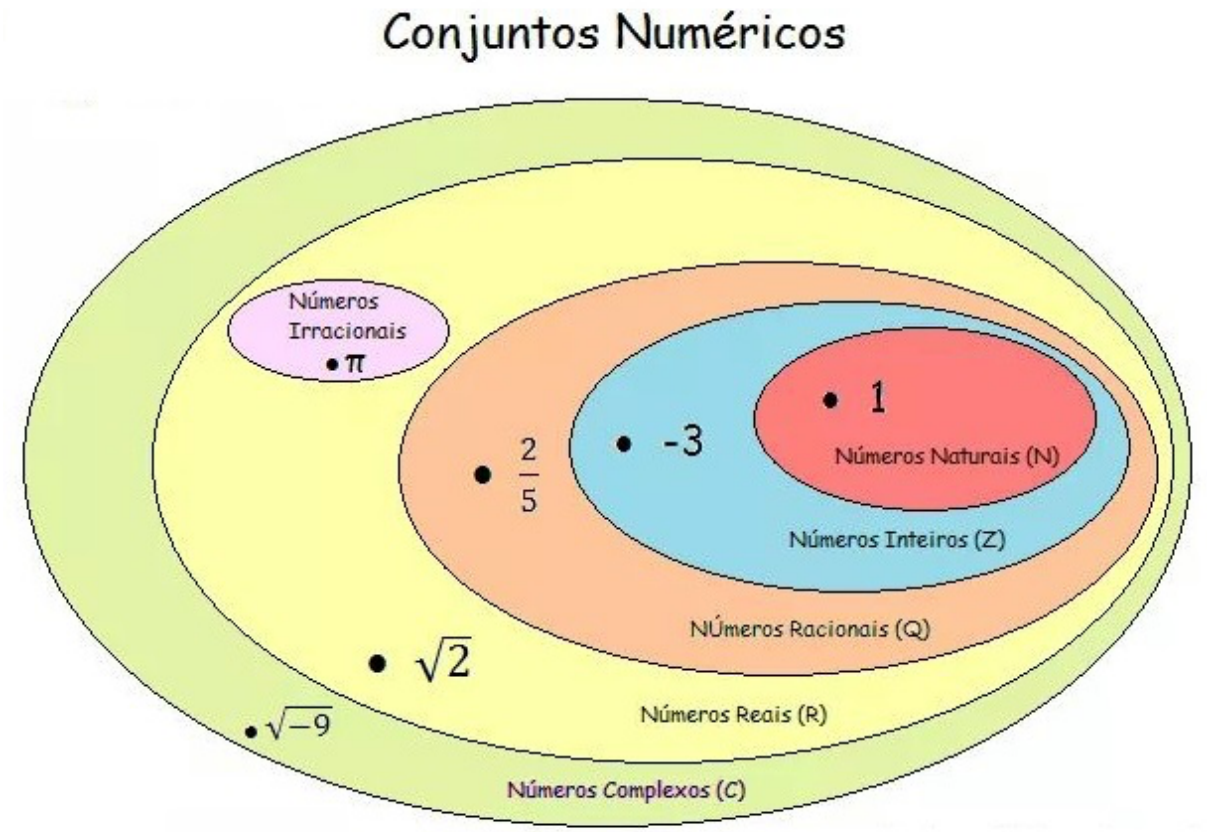
Sabemos, da Matemática Discreta e Teoria dos Números, que há a seguinte inclusão:



Refatore a classe Inteiro para utilizar o mecanismo de herança com a classe Racional.

EXERCÍCIO DE LABORATÓRIO

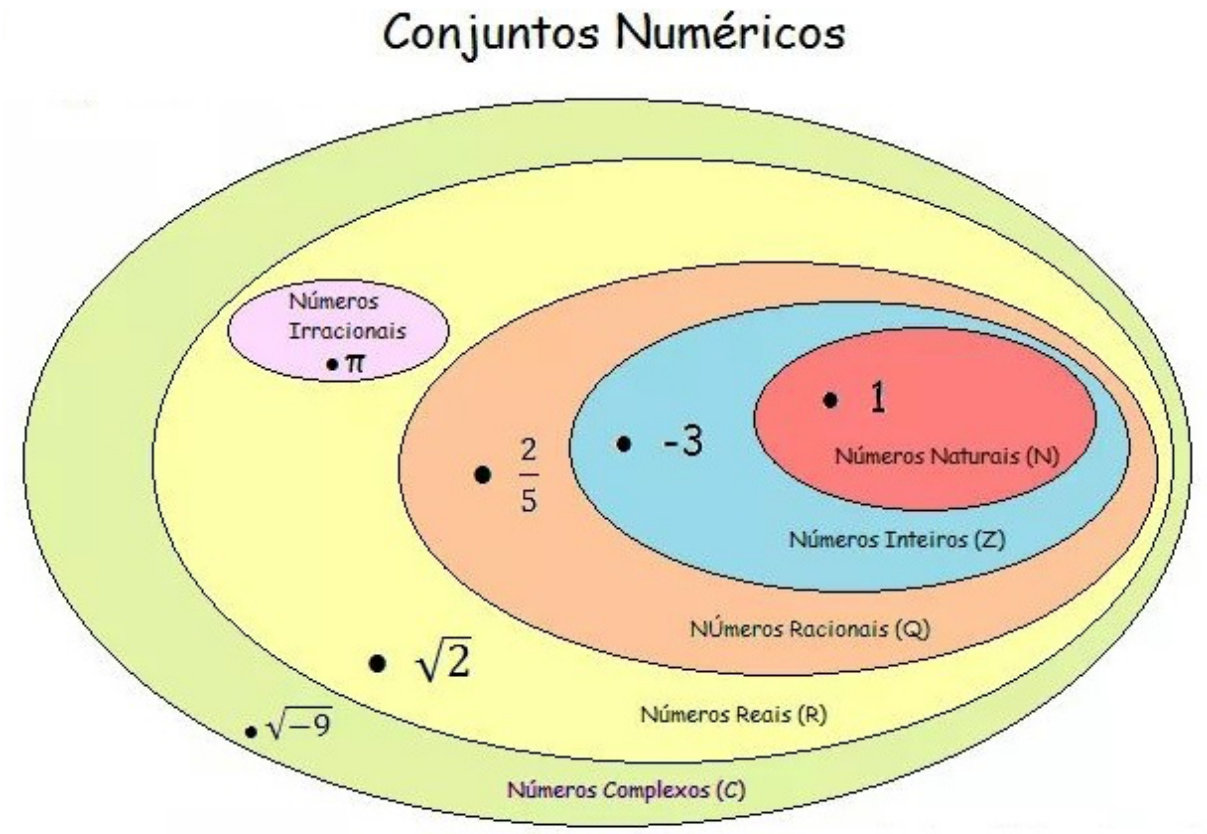
Sabemos, da Matemática Discreta e Teoria dos Números, que há a seguinte inclusão entre os conjuntos dos Números Racionais (Q) e dos Números Complexos (C):



Refatore a implementação da classe Racional para utilizar herança com a classe Complexo, implementada na última aula de laboratório.

EXERCÍCIOS EXTRA-CLASSE (ENTREGA MOODLE)

Considere, novamente, todos os conjuntos numéricos mostrados abaixo:



1. Implemente o tipo abstrato Real para reapresentar números reais.

2. Refatore a classe Racional para utilizar o mecanismo de herança com a classe Real.

3. Refatore a classe Real para utilizar o mecanismo de herança com a classe Complexo.