

DCC024 Linguagens de Programação  
2023.1

# **Tipos Abstratos de Dados**

## **Orientação a Objetos**

Haniel Barbosa



# Abstração

---

- ▷ Bárbara Liskov, recebedora do *Turing Award* em 2008, foi pioneira no uso de abstrações em linguagens de programação, nos anos 1970
- ▷ Um tipo abstrato de dado é definido por
  - ▶ operações
  - ▶ dados

The **use** which may be made of an abstraction is **relevant**.  
The **implemented** of the abstraction is **irrelevant**.

# Abstração

---

- ▷ Diferentes níveis de abstração facilitam *modularização*
  - ▶ Separar um problema em diferentes partes e entendê-las individualmente
    - Definições locais
  - ▶ Entender o relacionamento entre as diferentes partes
    - Contratos
  - ▶ Combinar as diferentes partes para a resolução do problema em questão

# Conjuntos em C

---

```
typedef struct  
{ ... } set;
```

```
void new(set* s);  
void add(set* s, unsigned e);  
void del(set* s, unsigned e);  
int contains(set* s, unsigned e);
```

# Conjuntos em C

---

```
typedef struct
{ ... } set;

void new(set* s);
void add(set* s, unsigned e);
void del(set* s, unsigned e);
int contains(set* s, unsigned e);

int main()
{
    set s; new(&s); add(&s, 2); add(&s, 3); add(&s, 5);
    printf("Contains %d? %d\n", 1, contains(&s, 1));
    printf("Contains %d? %d\n", 2, contains(&s, 2));
    printf("Contains %d? %d\n", 5, contains(&s, 5));
    del(&s, 5);
    printf("Contains %d? %d\n", 5, contains(&s, 5));
}
```

# Implementando conjuntos em C

---

```
typedef struct
{ unsigned* vector; unsigned size; unsigned capacity; } set;

void new(set* s)
{
    s->vector = (unsigned*) malloc(2 * sizeof(unsigned));
    s->size = 0; s->capacity = 2;
}

void add(set* s, unsigned e)
{
    if (contains(s, e)) return;
    if (s->size == s->capacity) {
        s->capacity *= 2;
        s->vector = realloc(s->vector, s->capacity * sizeof(int));
    }
    s->vector[s->size++] = e;
}
```

# Implementando conjuntos em C

---

```
void del(set* s, unsigned e)
{
    unsigned deleted = s->size;
    for (unsigned i = 0; i < s->size; ++i)
        if (s->vector[i] == e)
        {
            deleted = i; break;
        }
    if (deleted == s->size) return;
    for (unsigned i = deleted; i < s->size - 1; ++i)
        s->vector[i] = s->vector[i+1];
    s->size--;
}
```

```
int contains(set* s, unsigned e)
{
    for (unsigned i = 0; i < s->size; ++i)
        if (s->vector[i] == e) return 1;
    return 0;
}
```

# Implementando conjuntos em C *com bitsets*

---

```
typedef struct
{ unsigned* vector; unsigned capacity; } set;

#define INT_BITS 32

void new(set* s)
{
    s->vector = (unsigned*)malloc((1 + (60 / INT_BITS)) * sizeof(unsigned));
    s->capacity = 60;
}

void add(set* s, unsigned e)
{
    unsigned index = e / INT_BITS;
    unsigned offset = e % INT_BITS;
    unsigned bit = 1 << offset;
    s->vector[index] |= bit;
}
```



# Implementando conjuntos em C *com bitsets*

---

```
void del(set* s, unsigned e)
{
    unsigned index = e / INT_BITS;
    unsigned offset = e % INT_BITS;
    unsigned bit = 1 << offset;
    s->vector[index] &= ~bit;
}

int contains(set* s, unsigned e)
{
    unsigned index = e / INT_BITS;
    unsigned offset = e % INT_BITS;
    unsigned bit = 1 << offset;
    return s->vector[index] & bit;
}
```

```
int main()
{
    set s; new(&s);
    add(&s, 2); add(&s, 3); add(&s, 5);
    printf("Contains %d? %d\n", 1,
           contains(&s, 1));
    printf("Contains %d? %d\n", 2,
           contains(&s, 2));
    printf("Contains %d? %d\n", 3,
           contains(&s, 3));
    printf("Contains %d? %d\n", 5,
           contains(&s, 5));
    del(&s, 5);
    printf("Contains %d? %d\n", 5,
           contains(&s, 5));
}
```

# Implementando conjuntos em C *com bitsets*

---

```
int main()
{
    set s; new(&s);
    add(&s, 2); add(&s, 3); add(&s, 5);
    printf("Contains %d? %d\n", 1, contains(&s, 1));
    printf("Contains %d? %d\n", 2, contains(&s, 2));
    printf("Contains %d? %d\n", 3, contains(&s, 3));
    printf("Contains %d? %d\n", 5, contains(&s, 5));
    del(&s, 5);
    printf("Contains %d? %d\n", 5, contains(&s, 5));

    s.vector[0] = 16;
    printf("Contains %d? %d\n", 2, contains(&s, 2));
    printf("Contains %d? %d\n", 3, contains(&s, 3));
    printf("Contains %d? %d\n", 4, contains(&s, 4));
}
```

# Implementando conjuntos em SML com módulos

---

```
signature SET =  
sig  
  type set  
  val new : set  
  val add : set -> int -> set  
  val contains : set -> int -> bool  
  val remove : set -> int -> set  
end ;  
  
structure S = FunSet ;  
val s = S.new ;  
val s = S.add s 3 ;  
val contains1 = S.contains s 1 ;  
val contains3 = S.contains s 3 ;  
val s = S.remove s 3 ;  
val contains3 = S.contains s 3 ;
```

```
structure FunSet :> SET =  
struct  
  type set = int -> bool  
  val new = fn x => false  
  fun add s i = fn (x : int) => if x = i then true else s x  
  fun contains s i = s i  
  fun remove s i = fn (x : int) => if x = i then false else s x  
end;
```

# Implementando conjuntos em SML com módulos

---

Selamento opaco ( $:>$ ) ou transparente ( $:$ )

```
structure FunSet : SET =  
struct  
  type set = int -> bool  
  val new = fn x => false  
  fun add s i = fn (x : int) => if x = i then true else s x  
  fun contains s i = s i  
  fun remove s i = fn (x : int) => if x = i then false else s x  
end;
```

# Orientação a Objetos (OO)

---

Little bundles of **data** that know how to **do operations** on themselves.

- ▷ Objetos são valores de um tipo abstrato de dados
- ▷ Objetos se comunicam entre si através de suas operações
  - ▶ Chamadas de métodos vistas como *troca de mensagens*
- ▷ Uma forma comum de implementar objetos é via *instâncias* de *classes*

# Principais elementos de programação OO

---

## ▷ Encapsulamento

- ▶ Parte dos dados e operações de um objeto apenas não acessíveis ao objeto
- ▶ Comum separar partes públicas e privadas do estado e operações

# Principais elementos de programação OO

---

## ▷ Encapsulamento

- ▶ Parte dos dados e operações de um objeto apenas não acessíveis ao objeto
- ▶ Comum separar partes públicas e privadas do estado e operações

## ▷ Herança e subtipagem

- ▶ Definição de um tipo abstrato a partir de outro
- ▶ *Liskov's substitution principle*
  - Se  $S <: T$ , então objetos de tipo  $T$  podem ser substituídos por objetos de tipo  $S$ .

# Principais elementos de programação OO

---

## ▷ Encapsulamento

- ▶ Parte dos dados e operações de um objeto apenas não acessíveis ao objeto
- ▶ Comum separar partes públicas e privadas do estado e operações

## ▷ Herança e subtipagem

- ▶ Definição de um tipo abstrato a partir de outro
- ▶ *Liskov's substitution principle*
  - Se  $S <: T$ , então objetos de tipo  $T$  podem ser substituídos por objetos de tipo  $S$ .

## ▷ Polimorfismo

- ▶ Sobrecarga (*overloading*): Redefinição de operações *para* novos tipos
- ▶ Sobrescrita (*overriding*): Redefinição de operações *em* novos tipos
- ▶ *Method Dispatch*
  - Determinação de qual versão de uma operação invocar baseado nos parâmetros



# Exemplo de conjunto com bitsets em Python

---

```
INT_BITS = 32
```

```
def getIndex(element):
```

```
    index = int(element / INT_BITS)
```

```
    offset = element % INT_BITS
```

```
    bit = 1 << offset
```

```
    return (index, bit)
```

```
class Set:
```

```
    def __init__(self, capacity):
```

```
        self.vector = [0] * (1 + int(capacity / INT_BITS))
```

```
        self.capacity = capacity
```

```
    def add(self, element):
```

```
        (index, bit) = getIndex(element)
```

```
        self.vector[index] |= bit
```

```
    def delete(self, element):
```

```
        (index, bit) = getIndex(element)
```

```
        self.vector[index] &= ~bit
```

```
    def contains(self, element):
```

```
        (index, bit) = getIndex(element)
```

```
        return (self.vector[index] & bit) > 0
```

# Exemplo de conjunto com bitsets em Python

---

```
s = Set(60)
s.add(4)
s.contains(4)
s.delete(4)
s.contains(4)

s.vector[0] = 16
s.contains(4)
```

# Exemplo de conjunto com bitsets em Python

---

```
s = Set(60)
s.add(4)
s.contains(4)
s.delete(4)
s.contains(4)

s.vector[0] = 16
s.contains(4)

s.add(70)

def errorAdd(self, element):
    if (element > self.capacity):
        raise IndexError(str(element) + " is out of range.")
    else:
        (index, bit) = getIndex(element)
        self.vector[index] |= bit
        print(element, "added successfully!")

Set.add = errorAdd
```

# Exemplo de conjunto com bitsets em Python

---

```
s = Set(60)
```

```
s.add(4)
```

```
s.contains(4)
```

```
s.delete(4)
```

```
s.contains(4)
```

```
s.vector[0] = 16
```

```
s.contains(4)
```

```
s.add(70)
```

```
def errorAdd(self, element):
```

```
    if (element > self.capacity):
```

```
        raise IndexError(str(element) + " is out of range.")
```

```
    else:
```

```
        (index, bit) = getIndex(element)
```

```
        self.vector[index] |= bit
```

```
        print(element, "added successfully!")
```

```
Set.add = errorAdd
```

```
s = Set(0)
```

```
s.add(40)
```

```
s.add(70)
```

# Comparação entre diferentes linguagens

---

	Encapsulamento	Extensão
C	Não	Não
SML	Sim	Não
Python	Não	Sim
C++/Java/...	Sim	Sim

# Exemplo de conjunto com bitsets em Python

---

```
class ErrorSet(Set):
    def checkIndex(self, element):
        if (element > self.capacity):
            raise IndexError(str(element) + " is out of range.")

    def add(self, element):
        self.checkIndex(element)
        Set.add(self, element)
        print(element, "successfully added.")

    def delete(self, element):
        self.checkIndex(element)
        Set.delete(self, element)
        print(element, "successfully removed.")

    def contains(self, element):
        if element > self.capacity:
            return False
        else:
            return Set.contains(self, element)
```

# Classe para visita~ão em Python

---

```
class Visitor:
    "A parameterized list visitor."
    def __init__(self, cb):
        self.cb = cb
    def __str__(self):
        return "Visitor with callback: {0}".format(self.cb)
    def visit(self, n):
        for i in range(0, len(n)):
            n[i] = self.cb.update(n[i])
        return n

class CallbackBase:
    "The basic callback"
    def __init__(self):
        self.f = lambda x: x+1
    def __str__(self):
        return "basic callback"
    def shouldUpdate(self, i): return True
    def update(self, i):
        return self.f(i) if self.shouldUpdate(i) else i

l = [0, 1, 2, 3]; cb = CallbackBase()
v = Visitor(cb)
v.visit(l)
```

# Classe para visita~ão em Python

---

```
class CallbackEven(CallbackBase):
    def __str__(self):
        return "even callback"
    def shouldUpdate(self, i):
        return i % 2 == 0

v0 = Visitor(CallbackBase())
v1 = Visitor(CallbackEven())
v0.visit([0,1,2,3])
v1.visit([0,1,2,3])
```



# Classe para visita~ão em Python

---

```
class CallbackEven(CallbackBase):  
    def __str__(self):  
        return "even callback"  
    def shouldUpdate(self, i):  
        return i % 2 == 0
```

```
v0 = Visitor(CallbackBase())  
v1 = Visitor(CallbackEven())  
v0.visit([0,1,2,3])  
v1.visit([0,1,2,3])
```

```
class CallbackDouble(CallbackBase):  
    def __init__(self):  
        self.f = lambda x: 2*x  
    def __str__(self):  
        return "double callback"
```

```
v = Visitor(CallbackDouble())  
v.visit([0,1,2,3])
```

# Classe para visita~ão em Python

---

```
class CallbackComb(CallbackDouble, CallbackEven):
    def __str__(self):
        return "the combining callback"

v = Visitor(CallbackComb())
v.visit([0,1,2,3])
```

# Classe para visita~ão em Python

---

```
class CallbackComb(CallbackDouble, CallbackEven):  
    def __str__(self):  
        return "the combining callback"
```

```
v = Visitor(CallbackComb())  
v.visit([0,1,2,3])
```

```
class CallbackOtherComb(CallbackDouble, CallbackEven):  
    def update(self, i):  
        return self.f(self.f(i)) if self.shouldUpdate(i) else i
```

```
v = Visitor(CallbackOtherComb())  
v.visit([0,1,2,3])  
print(v)
```

# Ligação estática e dinâmica em C++

---

```
class A
{
public:
    void f () { std::cout << "A!\n"; }
};
```

```
class B
{
public:
    void f () { std::cout << "B!\n"; }
};
```

```
class C : public A, public B
{
};
```

```
int main()
{
    C* pc = new C;
    pc->f ();
}
```

# Ligação estática e dinâmica em C++

---

```
class A
{
public:
    void f () { std::cout << "A!\n"; }
};
```

```
class B
{
public:
    void f () { std::cout << "B!\n"; }
};
```

```
class C : public A, public B
{
};
```

```
int main()
{
    C* pc = new C;
    pc->A::f ();
    pc->B::f ();
}
```

# Ligação estática e dinâmica em C++

---

```
class A
{
public:
    void f () { std::cout << "A!\n"; }
};
```

```
class B
{
public:
    void f () { std::cout << "B!\n"; }
};
```

```
class C : public A, public B
{
    void f () { std::cout << "C!\n"; }
};
```

```
int main()
{
    C* pc = new C;
    pc->f ();
}
```

# Ligação estática e dinâmica em C++

---

```
void aux(A* p)
{
    p->f();
}
```

```
int main()
{
    C* pc = new C;
    pc->f();

    aux(pc);
}
```

# Ligação estática e dinâmica em C++

---

```
class A
{
public:
    virtual void vf() { std::cout << "A!\n"; }
};

class B
{
public:
    virtual void vf() { std::cout << "B!\n"; }
};

class C : public A, public B
{
    void vf() override { std::cout << "C!\n"; }
};
```

```
void aux(A* p)
{
    p->vf();
}
```

```
int main()
{
    C* pc = new C;
    aux(pc);
}
```