Introdução

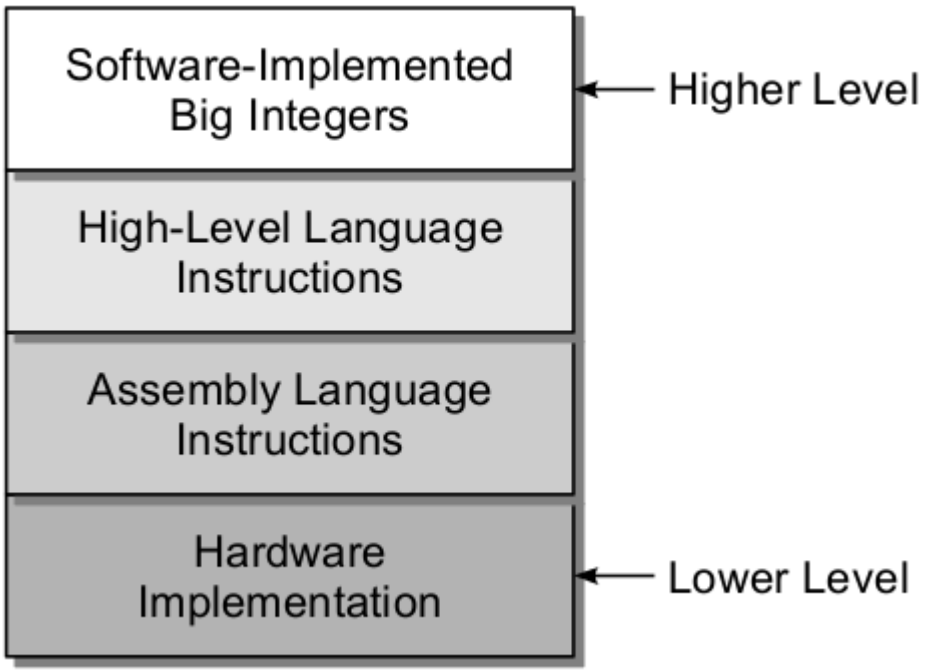
As linguagens de programação oferecem tipos de dados como uma parte integrante da linguagem:

- Estes tipos de dados são conhecidos como **tipo primitivos de dados**.
- Esses tipos podem ser: simples ou complexos.
- Inteiros e reais são tipos simples.
- Os tipos complexos são construídos a partir de múltiplos tipos primitivos ou mesmo com outros tipos de dados complexos.
- Em Python, objetos, strings, listas e dicionários são exemplos de tipos complexos.
- Muitas vezes, os tipos primitivos oferecidos pelas linguagens de programação não são suficientes para resolver problemas maiores ou mais complexos.
- Portanto, a maioria das linguagens de programação oferece recursos para que os programadores criem os seus próprios tipos de dados.

Abstração

É um mecanismo para separar as propriedades de um objeto e restringir o foco para o que seja realmente relevante.

Exemplo de diversos níveis de abstração com aritmética de inteiros:



Tipo Abstrato de Dados (TAD)

- Ou **Abstract Data Type (ADT)**.
- É um tipo de dados definido pelo programador que especifica um conjunto de valores de dados e uma coleção bem definida de operações que podem ser executadas nesses valores.
- TAD são definidos de maneira independente da sua implementação.
- A interação com um TAD é realizado através da sua **interface** ou pelo seu conjunto de funções.
- A consequência disso é o **ocultamento de informação** (*information hiding*).

Um TAD é como uma caixa-preta:

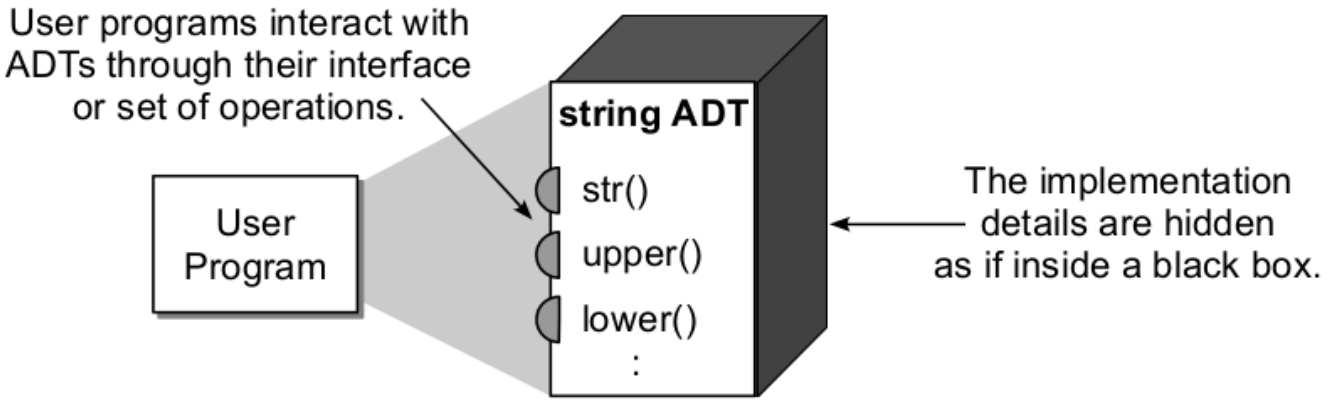


Figure 1.2: Separating the ADT definition from its implementation.

O conjunto de operações pode ser agrupado em quatro categorias:

- **Construtores (Constructor)**: criam e inicializam novas instâncias do TAD.
- **Acessores (Accessor)**: Retorna os dados contidos em uma instância sem modificá-la.
- **Modificadores (Mutator)**: Modifica o conteúdo de uma instância de um TAD.
- **Iteradores**: Processa os dados individuais dos componentes.

Python Construtor

- Em Python, um construtor é um método especial usado para inicializar as instâncias de uma classe.
- Os construtores podem ser de dois tipos:

    - Construtor parametrizado.
    - Construtor não-parametrizado.

- O construtor é executado quando nós criamos o objeto de uma classe.
- Os construtores também verificam se existem os recursos necessários para que o objeto execute qualquer das suas tarefas.
- O método **init** simula o construtor de uma classe.
- Esse método é chamado quando a classe é instanciada.
- Esse método é comumente usado para inicializar os atributos da classe.
- Toda classe deve ter um construtor, mesmo se ele simplesmente confiar no construtor padrão (*default*).

Exemplo de **construtor** em Python (Exemplo 1)

```
class Employee:
  # Método construtor
  def __init__(self, name, id):
    self.id = id
    self.name = name

  def display (self):
    print ("ID: %d \nName: %s"%(self.id, self.name))
```

Instanciação da classe

```
emp1 = Employee("John", 101)
emp2 = Employee("David", 102)
```

Execução de método interno

```
emp1.display()
emp2.display()
```

```
ID: 101
Name: John
ID: 102
Name: David
```

Exemplo de programa para contar o número de objetos de uma classe (Exemplo 2)

```
```

```
class Student:
  count = 0
  def __init__(self):
    Student.count = Student.count + 1

s1 = Student()
s2 = Student()
s3 = Student()

print ("O número de estudantes é:", Student.count)
```

```
O número de estudantes é: 3
```

Exemplo de construtor não-parametrizado (Exemplo 3)

```
class Student:
  #Construtor sem parâmetros
  def __init__(self):
    print ('Este método não é parametrizado.')
  def show (self, name):
    print ("Hello", name)

student = Student()
student.show("John")
```

```
Este método não é parametrizado.
Hello John
```

Exemplo de construtor parametrizado (Exemplo 4)

```
class Student:
  #Construtor com parâmetros
  def __init__(self, name):
    print ('Este método é parametrizado.')
    self.name = name
  def show (self):
    print ("Hello", self.name)

student = Student("John")
student.show()
```

```
Este método é parametrizado.
Hello John
```

Funções *built-in*

SN	Function	Description

SN	Function	Description
1	getattr(obj,name,default)	It is used to access the attribute of the object.
2	setattr(obj, name,value)	It is used to set a particular value to the specific attribute of an object.
3	delattr(obj, name)	It is used to delete a specific attribute.
4	hasattr(obj, name)	It returns true if the object contains some specific attribute.

Exemplo:

```
class Student:
  def __init__(self, name, id, age):
    self.name = name
    self.id = id
    self.age = age
```

Cria um objeto da classe

```
s = Student ("John", 101, 22)
```

Imprime o nome do atributos do objeto `s`

```
print (getattr(s, 'age'))
```

```
22
```

Atualiza o valor do atributo `age` para 23

```
setattr(s, "age", 23)
```

Imprime o valor modificado de `age`

```
print (getattr(s, 'age'))
```

```
23
```

Imprime `True` se o estudante possui um atributo `id`

```
print (hasattr(s, 'id'))
```

```
print (hasattr(s, 'id'))
```

```
True
```

Apaga o atributo `age`

```
delattr(s, 'age')
```

Aparecerá um erro porque o atributo foi removido anteriormente:

```
print (s.age)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call
last)
<ipython-input-19-e200e346a988> in <module>()
----> 1 print (s.age)

AttributeError: 'Student' object has no attribute 'age'
```

SEARCH STACK OVERFLOW

Outras funções *built-in* em Python:

SN	Attribute	Description
1	__dict__	It provides the dictionary containing the information about the class namespace.
2	__doc__	It contains a string which has the class documentation
3	__name__	It is used to access the class name.
4	__module__	It is used to access the module in which, this class is defined.
5	__bases__	It contains a tuple including all base classes.

Python Accessor/Mutator

Python Accessor:

- Um método accessor retorna a informação sobre o objeto, mas não muda o estado ou o objeto.
- Esse método normalmente é usado com a palavra get.

Python Mutator:

- Um método mutator é uma função que modifica a variável interna de alguma maneira.
- A mais simples forma de um mutator é atribuir um novo valor para uma variável.
- Esse método normalmente é usado com a palavra **set**.

Exemplo:

```
class P:
  def __init__(self, x):
    self.__x = x

  def get_x(self):
    return self.__x

  def set_x(self, x):
    self.__x = x
```

```
p1 = P(42)
p2 = P(4711)

print (p1.get_x())
```

```
42
```

```
p1.set_x(47)
p1.set_x(p1.get_x() + p2.get_x())

print (p1.get_x())
```

```
4758
```

Observe que o encapsulamento em Python é questionável!

Python Accessor/Mutator (Exemplo 2)

```
class P:
  def __init__(self, x):
    self.x = x
```

```
p1 = P(42)
p2 = P(4711)
print (p1.x)
```

```
42
```

```
p1.x = 47
p1.x = p1.x + p2.x
print (p1.x)
```

```
4758
```

Iterators

Um iterator pode ser visualizado como um ponteiro para um container, isto é, uma estrutura do tipo lista que pode percorrer sobre todos os elementos deste container.

Exemplo de iterators:

```
cities = ["Paris","Berlin","Frankfurt"]
for location in cities:
  print("location: " + location)
```

```
location: Paris
location: Berlin
location: Frankfurt
```

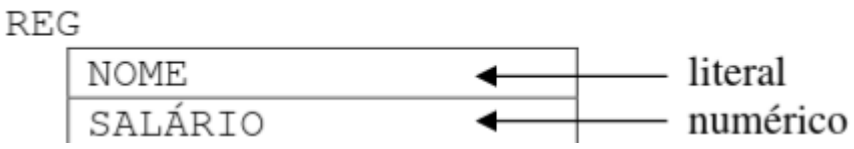
Tipo Abstrato de Dados (TADS)

Existem algumas vantagens no uso de TADs:

- Foco na resolução do problema ao invés de se preocupar com detalhes de implementação.
- A implementação do TAD pode ser modificada sem ter a necessidade de modificar o programa que utiliza o TAD.
- É mais fácil gerenciar e dividir programas grandes (reais) em módulos menores.

Exercícios

1. Explique a vantagem do uso de tipos abstratos de dados. Dê um exemplo.
2. Crie um TAD em Python para o elemento abaixo:



3. Um número complexo é representado como x + i.y, onde i2 = -1, sendo x a sua parte real e y a sua parte imaginária. Ambas são representadas por números reais. Crie um TAD em Python que represente os números complexos com as seguintes funções: a) Criar um número complexo. b) Destruir um número complexo. c) Soma de dois números complexos. d) Subtração de dois números complexos.

Referências

[Java Point]