04834580 Software Engineering (Honor Track) 2024-25

Abstraction Mechanisms

Sergey Mechtaev
mechtaev@pku.edu.cn
School of Computer Science, Peking University

Definition ([1])

- The act or process of leaving out of consideration one or more properties of a complex object so as to attend to others.
- A general concept formed by extracting common features from specific examples.

From "The Preparation of Programs for an Electronic Digital Computer" (1957) on "automatic programming" [2]:

> *There has been much controversy [...] Some went so far as to assert that programmers should be compelled to write order in a form as near as possible to that which they take inside the machine, and that attempts to make the machine assist with the clerical tasks of programming lead only to a wasteful dissipation of effort.*

	T	200	K	
	T		F	Clear $0F$ initially
	S	5*		Set count in $4F$
L3*	T	4	F	
	A	1*		Increase address in A-order
	A	2	F	*Note:* $C(2F) = P\ 1\ F$.
	T	1*		
	A		F	
L1*	(A	399	F)	Add contribution to sum in $0F$
	T		F	
	A	4	F	
	A	2	F	Count
	G	3*		
	H		F	
L6*	A	6*		Print
	F	50*		
	Z		F	Stop
L5*	‖ P	100	F	

北京大学

```scheme
(define (sum-integers a b)
  (if (> a b)
    0
    (+ a (sum-integers (+ a 1) b))))

(define (sum-cubes a b)
  (if (> a b)
    0
    (+ (cube a) (sum-cubes (+ a 1) b))))

(define (pi-sum a b)
  (if (> a b)
    0
    (+ (/ 1.0 (* a (+ a 2)))
       (pi-sum (+ a 4) b))))
```

```scheme
(define ( <name> a b)
  (if (> a b)
    0
    (+ ( <term> a)
       ( <name> ( <next> a) b))))
```

Procedural Abstraction

Each abstraction layer performs one meaningful task; each layer hides the implementation details of the layer below.

```scheme
(define (sum term a next b)
  (if (> a b)
    0
    (+ (term a)
       (sum term (next a) next b))))
```

```scheme
(define (identity x) x)
(define (cube x) (* x x x))
(define (inc x) (+ x 1))
(define (pi-term x) (/ 1.0 (* x (+ x 2))))
(define (pi-next x) (+ x 4))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (sum-integers a b)
  (sum identity a inc b))
(define (sum-cubes a b)
  (sum cube a inc b))
(define (pi-sum a b)
  (sum pi-term a pi-next b))
```

Dynamic Dispatch

北京大学

Definition

Dynamic dispatch is the process of selecting which implementation of a polymorphic operation (method or function) to call at run time.

This code violates the Open-Closed Principle:

```
def calculate_area(shape):
    if shape['type'] == 'circle':
        return math.pi * (shape['radius'] ** 2)
    elif shape['type'] == 'rectangle':
        return shape['width'] * shape['height']
    else:
        raise ValueError("Unknown shape type")

circle = {'type': 'circle', 'radius': 5}
rectangle = {'type': 'rectangle', 'width': 4, 'height': 6}

print("Circle Area:", calculate_area(circle))
print("Rectangle Area:", calculate_area(rectangle))
```

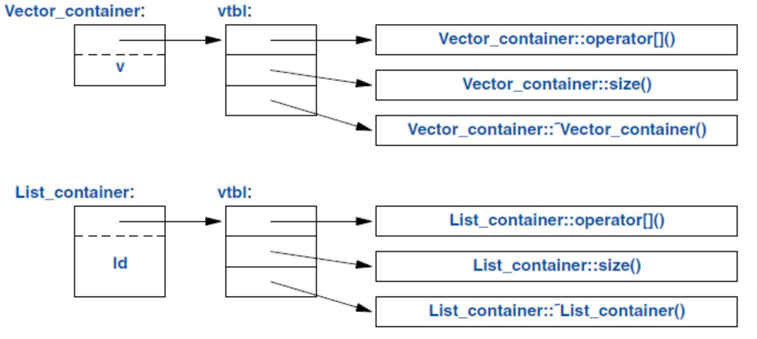
Extracting the polymorphic operation:

```
def calculate_circle_area(circle):
    return math.pi * (circle['radius'] ** 2)

def calculate_rectangle_area(rectangle):
    return rectangle['width'] * rectangle['height']

def calculate_area(shape):
    if shape['type'] == 'circle':
        return calculate_circle_area(shape)
    elif shape['type'] == 'rectangle':
        return calculate_rectangle_area(shape)
    else:
        raise ValueError("Unknown shape type")

circle = {'type': 'circle', 'radius': 5}
rectangle = {'type': 'rectangle', 'width': 4, 'height': 6}

print("Circle Area:", calculate_area(circle))
print("Rectangle Area:", calculate_area(rectangle))
```

```python
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return math.pi * (self.radius ** 2)

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height
    def area(self):
        return self.width * self.height

class Triangle(Shape):
    def __init__(self, base, height):
        self.base = base
        self.height = height
    def area(self):
        return 0.5 * self.base * self.height
```

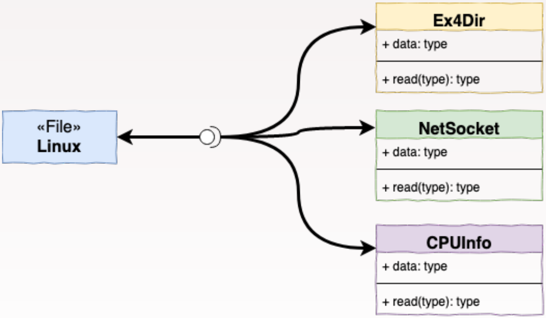
```python
shapes = [
    Circle(radius=5),
    Rectangle(width=4, height=6),
    Triangle(base=3, height=4) ]

for shape in shapes:
    print(f"Area: {shape.area()}")
```

```
class Container {
public:
    virtual double& operator[](int) = 0;
    virtual int size() const = 0;
    virtual ~Container() {}
};
```



Dynamic Dispatch in Linux I/O

```
int open(const char* path, int flags, mode_t permissions)
int close(int fd)
ssize_t read(int fd, void* buffer, size_t const)
ssize_t write(int fd, const void* buffer, size_t count)
off_t lseek(int fd, off_t offset, int referencePosition)
```



Dynamic Dispatch in Linux I/O

北京大学

A generic struct is used to store pointers to implementation of I/O functions:

```
struct file_operations {
  struct module *owner;
  loff_t (*llseek) (struct file *, loff_t, int);
  ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
  ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
   //...
}
```

An implementation struct is filled with concrete code:

```
const struct file_operations ext4_dir_operations = {
  .llseek        = ext4_dir_llseek,
  .read        = generic_read_dir,
  //...
};
```

Cost of Abstraction

A program in Java:

```java
private static double average(int[] data) {
  int sum = 0;
  for (int i = 0; i < data.length; i++) {
    sum += data[i];
  }
  return sum * 1.0d / data.length;
}
```

Time: **30 ms**

The same program using Scala abstractions:

```scala
def average(x: Array[Int]): Double = {
  x.reduce(_ + _) * 1.0 / x.size
}
```

Time: **518 ms**

Leaky Abstractions

Joel Spolsky: "All non-trivial abstractions, to some degree, are leaky." [3]

```cpp
#include <iostream>
#include <string>

int main() {
    std::string str1 = "Hello";
    std::string result1 = str1 + ", World!";
    std::cout << "result1: " << result1 << std::endl;

    // ERROR: Invalid operands to binary +
    std::string result2 = "Hello" + ", World!";

    return 0;
}
```

A program with Scala abstractions:

```
def average(x: Array[Int]): Double = {
  x.reduce(_ + _) * 1.0 / x.size
}
```

Time: **518 ms**

The same program in Rust:

```
fn average(xs: &[i32]) -> f64 {
  xs.iter().fold(0, |x, y| x + y) as f64 / xs.len() as f64
}
```

Time: **18 ms**

北京大学

A polymorphic operation implementing a trait:

```
trait Foo {
    fn method(&self) -> String;
}
impl Foo for u8 {
    fn method(&self) -> String { format!("u8: {}", *self) }
}

impl Foo for String {
    fn method(&self) -> String { format!("string: {}", *self) }
}
```

Rust Static Dispatch

The operation is chosen at compile time:

```rust
fn do_something<T: Foo>(x: T) {
    x.method();
}

fn main() {
    let x = 5u8;
    let y = "Hello".to_string();

    do_something(x);
    do_something(y);
}
```

Rust Dynamic Dispatch With Trait Objects

The operation is chosen at runtime:

```rust
fn main() {
    let items: Vec<Box<dyn Foo>> = vec![
        Box::new(5u8),
        Box::new(String::from("Hello")),
    ];

    for item in items {
        item.method();
    }
}
```

Overloading in Java & C++

```
interface Animal {
    void makeSound();
}
class Dog implements Animal {
    void makeSound() {
        System.out.println("Woof! Woof!");
    }
}
class Cat implements Animal {
    void makeSound() {
        System.out.println("Meow! Meow!");
    }
}
class SoundPlayer {
    void playSound(Dog dog) {
        dog.makeSound();
    }
    void playSound(Cat cat) {
        cat.makeSound();
    }
}
```

```
List<Animal> animals = new ArrayList<>();
animals.add(new Dog());
animals.add(new Cat());
SoundPlayer soundPlayer = new SoundPlayer();

for (Animal animal : animals) {
  animal.makeSound();
  soundPlayer.playSound(animal);
}
```

What is the difference?

[1] Jeff Kramer.
Is abstraction the key to computing?
Communications of the ACM, 50(4):36–42, 2007.

[2] Maurice V Wilkes, David J Wheeler, Stanley Gill, and FJ Corbató.
The preparation of programs for an electronic digital computer, 1958.

[3] Joel Spolsky.
The law of leaky abstractions.
`https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/`, 2002.