Conceptos de POO

Ana Lilia LAUREANO CRUCES

UAM-A

Tipo Clase

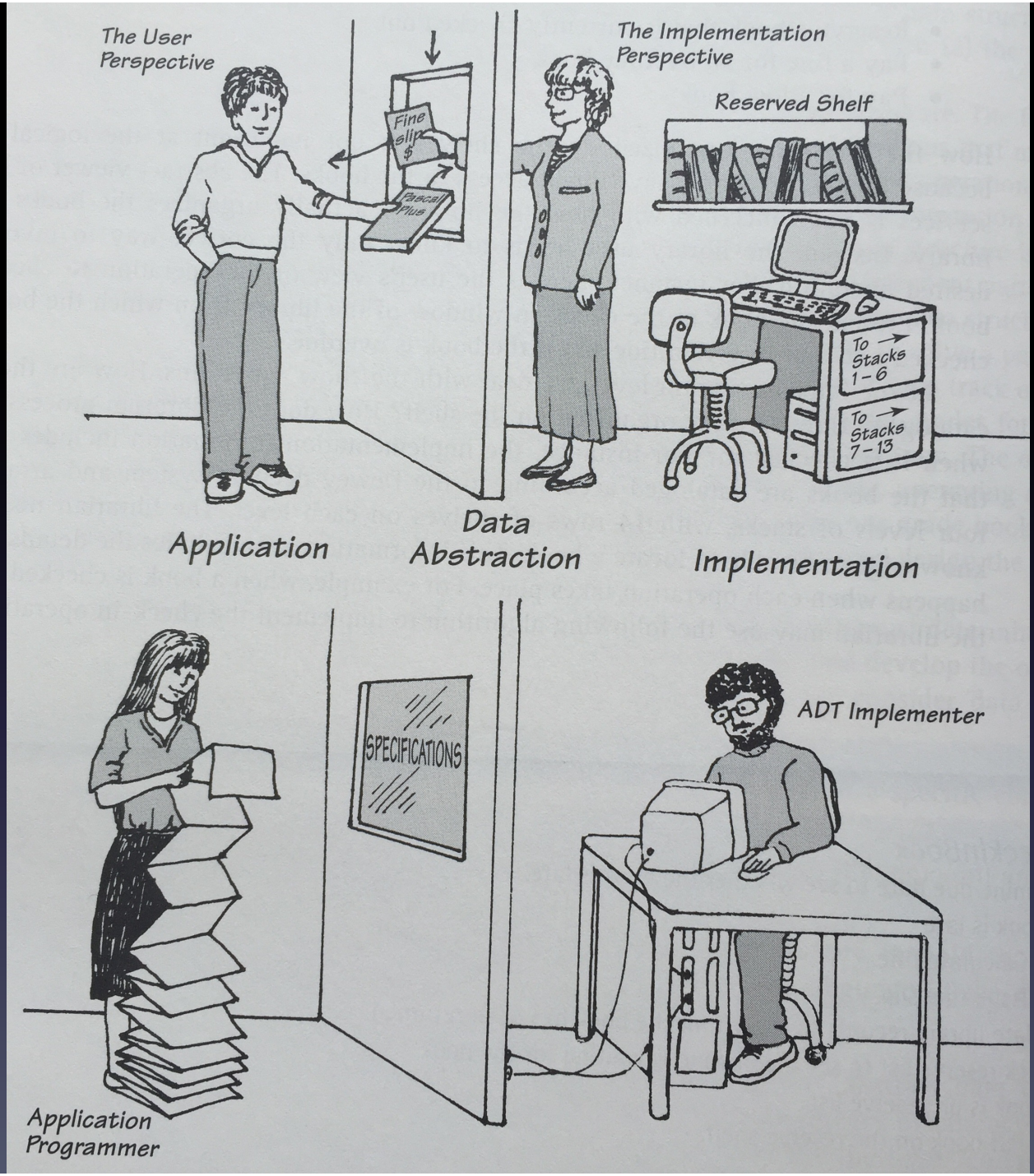
- Clase:
  - Es un tipo no estructurado que encapsula un numero fijo de numero de datos con las funciones que los manipulan.
  - Las funciones predefinidas sobre una instancia de clase; son todas asignadas y accesdas en en el componente.

- Cliente:
  - Es software que declara y manipula objetos (instancias) de una clase particular.

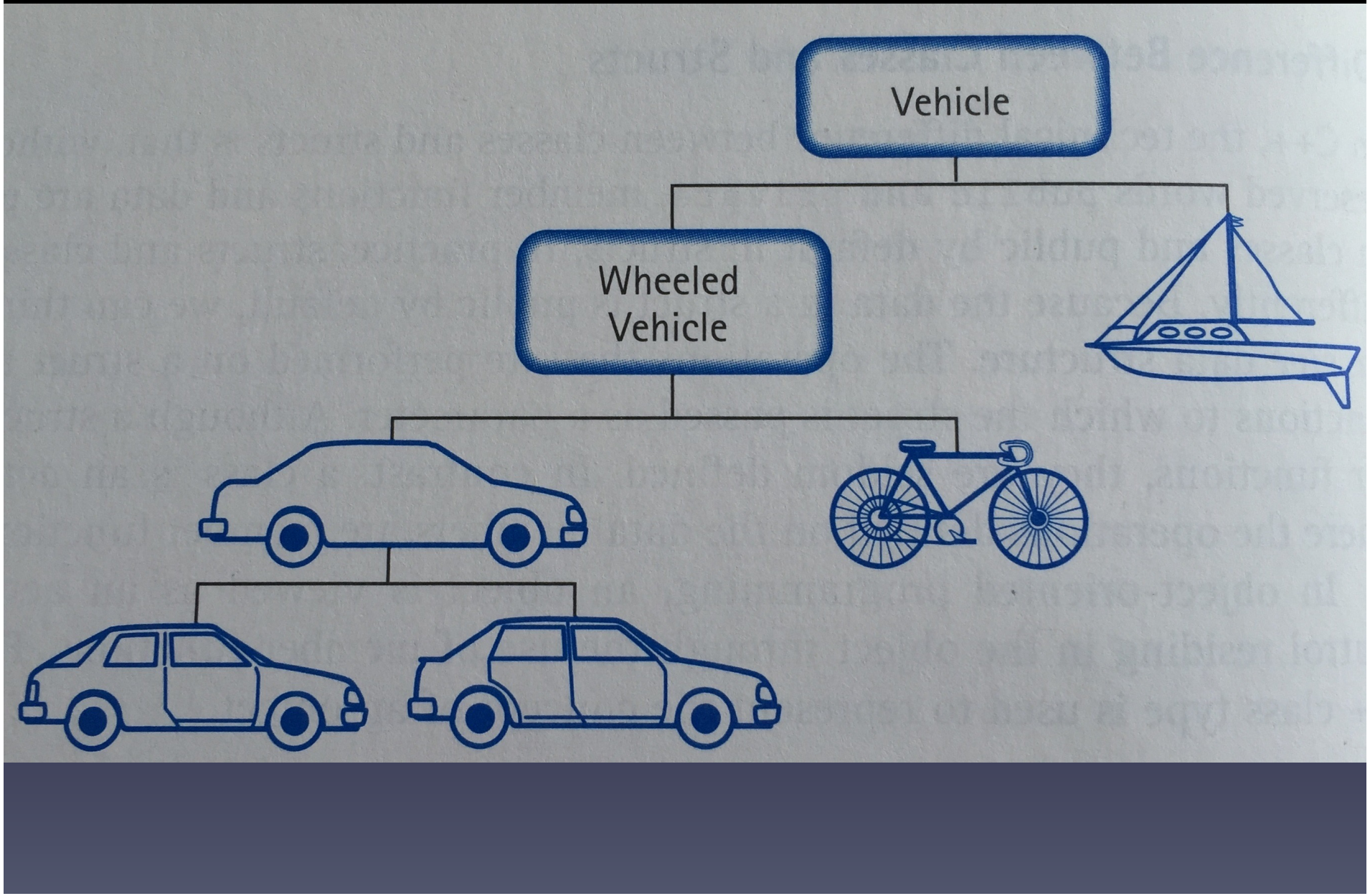
Existen tres conceptos asociados

- Abstracción de Datos:
  - La separación de las propiedades lógicas; del tipo de dato de su implementación

- Encapsulamiento de datos:
  - la separación de la representación de los datos de la aplicación que utiliza el dato en el nivel lógico
  - Un aspecto del programa que fuerza al ocultamiento de esta última.


The User Perspective
The Implementation Perspective
Reserved Shelf
Fine slip $
Pascal Plus
To Stacks 1-6
To Stacks 7-13
Application
Data Abstraction
Implementation
SPECIFICATIONS
ADT Implementer
Application Programmer

- Herencia:
  - Un mecanismo utlizado a través de una jerarquía de clases, en la que cada descendiente hereda las propiedades (datos y operaciones) de su clase ancestro.


Vehicle
Wheeled Vehicle

Polimorfismo compuesto por:

- Clase Base:
  - La calse heredada
- Clase Derivada:
  - la clase que hereda
- Polimorfismo:
  - La habilidad de determinar cual dadas varias operaciones con el mismo nombre es la apropiada. El ligado se hace con una combinación estática y dinámica.

- Sobrecarga:
  - Dado el mismo nombre a más de una función o el mismo símbolo de operador. Generalmente se asocia a un ligado estático.
- Tiempo de ligado:
  - El tiempo a través del cual un nombre o símbolo es asociado al código apropiado.

- Tiempo de Ligado Estático:
  - Se determina en tiempo de compilación
- Tiempo de Ligado Dinámico:
  - Se pospone la decisión hasta el tiempo de ejecución.

Clase DateType (definición/Inerfaz = .h)

```
// Declaramos una clase que representa al TAD Date
// este archivo debe ser un archivo Date.h
#ifndef _DATETYPE_H_
#define _DATETYPE_H_
enum RelationType {LESS, EQUAL, GREATER};
class DateType
{
 public:
  void Initialize (int newMonth, int newDay, int newYear);
  int GetYear () const;
  int GetMonth () const;
  int GetDay () const;
  RelationType ComparedTo (DateType  someDate);
 private:
  int year;
  int month;
  int day;
};
#endif /* _DATETYPE_H_ */
```

Clase DateType (Implementación(Cómo) =.CPP

```
#include "DateType.h"
void DateType::Initialize (int newMonth, int newDay, int newYear) {
 year = newYear;
 month = newMonth;
 day = newDay;
}
int DateType::GetMonth () const
{
 return month;
}
int DateType::GetYear () const
 {
 return year;
}
int DateType::GetDay () const
{
 return day;
}
```

Clase DateType (Implementación(Cómo) = .CPP

```
RelationType DateType::ComparedTo (DateType aDate)
{
 if (year < aDate.year)
   return LESS;
 else if (year > aDate.year)
   return GREATER;
 else if (month < aDate.month)
   return LESS;
 else if (month > aDate.month)
   return GREATER;
 else if (day < aDate.day)
   return LESS;
 else if (day > aDate.day)
   return GREATER;
 else
   return EQUAL;
}
```

Cliente

```
#include <iostream>
#include "DateType.h"
using namespace std;
void main ()
{
 DateType today, anotherDay;;
 today.Initialize (9, 24, 2003);
 anotherDay.Initialize (9, 25, 2003);
 cout << "Today is " << today.GetMonth() << "/" << today.GetDay() << "/" << today.GetYear() << endl;
 cout << "Another day is " << anotherDay.GetMonth() << "/" << anotherDay.GetDay() << "/" << anotherDay.GetYear() << endl;
 switch (today.ComparedTo (anotherDay))
{
 case LESS :
  cout << "today comes before anotherDay" << endl;
  break;
 case GREATER :
  cout << "today comes after anotherDay" << endl;
  break;
 case EQUAL :
  cout << "today and anotherDay are the same" << endl;
  break;
 }
}
```

Un objeto tiene 3 tipos de relaciones

- Es independiente con respecto a las demás
- Esta relacionado con otra a través de una composición (o contenida)
- Esta relacionada a través de la herencia

Relación a través de una Composición o Contenida

- Es cuando una clase contiene a otra clase como miembro de sus datos.

```
#include <string>
#include <DateType.h>
class PersonType
{
public:
        void Initialize (string, DateType);
        string GetName ( ) const;
        DateType GetBirthDate ( ) const; // Contenido o Composición
private:
        string name;
        DateType birthday;
}
```

Relación a través de: herencia o derivación

```
// Clase Base
#include <string>
class MoneyType
{
public:
        void Initialize (long, long);
        long GetDollars ( ) const;
        long GetCents ( ) const;
private:
        long dollars, cents;
};
```

```
//clase derivada
#include <string>
#include "MoneyType.h"

class ExtMoneyType : public MoneyType // herencia de la clase base MoneyType
{
public:
        string GetCurrency ( ) const;
        void Initialize (long, long, string);

private:
        string currency;
};
```

Clase ExtMoneyType (Implementación(Cómo) =.CPP

- // implementación de ExtMoney.h
- #include <string>
- #include "MoneyType"
- void ExtMoneyType::Initialize (long new cents, string new currency)
- {
- currency = new currency;
- Money :: Initialize (newDollars, newCents);
- }
- string ExtMoneyType :: GetCurrency ( ) const
- {
- return currency;
- }

Cliente para ExtMoneyType

```
// Cliente de la clase ExtMoneyType
#include <iostream>
#include "ExtMoneyType.h"
using namespace std;

 void main()
{
        ExtMoneyType extMoney:
        MoneyType money;

        money.Initialize (20,66);
        extMoney.Initialize (20, 66, "francos");
}
```