Capítulo 1
Conceptos básicos: clases y objetos

Universidad de Chile

Departamento de Cs. De la Computación

Prof.: Nancy Hitschfeld Kahler

Contenido

- Clases y objetos
- Objetos: estado, comportamiento e identidad
- Tipos de datos abstractos (TDAs)
- De TDAs a clases
- Ejemplo de TDA genérico en C++
- Ejemplo de TDA genérico en Java

Clases y Objetos

- **Objeto:**
  - *elemento que surge del análisis del dominio del problema a resolver*
  - *instancia de una clase (programación)*
- **Clase:**
  - *tipo de molde o plantilla que define los valores que almacena un objeto en sus variables de instancia, y acciones u operaciones que se le puede aplicar a través de sus métodos*

Clases y objetos

- Puede constar de:
  - Variables de la clase
  - Variables instancia
  - Métodos
- Clase en Java:

```
public class Punto {
        private int x;   // variable  de instancia
        private int y;
        public Punto(int _x, int _y){ x=_x;  y= _y;}
        public int getX(){ return x; }  // método
        public int getY(){ return y; }
}
```


Clases y objetos

- **Declaración y uso de un objeto de la clase Punto**
  - Punto cero = new Punto(0,0);
  - int valor_x = cero.getX();
- **Características de un objeto**
  - estado: definido a través de las variables de instancia
  - comportamiento: definido a través de las operaciones o métodos
  - identidad: es lo que se preserva a pesar que el estado cambie

Objetos: definición de clases (c++)

```
Class Punto{
     int x,y;
  public:
    Punto(){ x=0; y=0; }
    Punto(int _x,int _y) {
         x = _x; y = _y;
    }

}
Class Color{
     float  rgb[3];
  public:
     Color(){ rgb[0] = 1;
              rgb[1] = rgb[2] = 0;
      }
    Color(float r, float g,
          float b);

}
```

```
Class Figura{
       Punto centro;
       Color color;

   public:
        Figura();
        Figura(Punto centro,
               Color _color);
        void mover(Punto hacia);
        Punto donde();

}
```


Objetos: declaración, estado y comportamiento

Uso:

```
Color azul(0,0,1);
Punto posicion(1,1);
Punto otra_posicion(2,3);
Figura rectangulo;
Figura circulo(posicion,azul);
rectangulo.mover(otra_posicion);
...
posicion = rectangulo.donde();
```

Objetos:

- azul
- posicion
- otra_posicion,
- rectangulo
- circulo

Estado:

```
azul: (0,0,1)
posicion: (1,1)
otra_posicion: (2,3)
rectangulo: (0,0) (1,0,0)
circulo: (1,1) (0,0,1)
rectangulo: (2,3) (1,0,0)
...
posicion: (2,3)
```

Comportamiento:

- rectangulo:
  - mover
  - donde

Objetos: identidad

C++

```
Punto p1(1,2);
Punto *p2 = new Punto(3,4);
Punto *p3 = new Punto(1,2);
Punto *p4;
p4 = p3;
```

p1, *p2 y *p3 son objetos distintos
*p4 y *p3 son el mismo objeto

Java

```
Punto p1 = new Punto(1,2);
Punto p2 = new Punto(3,4);
Punto p3 = new Punto(1,2);
Punto p4 = p3;
```

p1, p2 y p3 son objetos distintos
p3 y p4 son el mismo objeto

Tipos de datos abstractos (TDA)

- **¿Cómo definir buenos objetos?** Con un buen TDA.
- **¿Qué característica debe cumplir un buen TDA?**
- **Su descripción debe ser:**
  - precisa y no ambigua
  - completa
  - no sobre-especificada
  - independiente de la implementación

Tipos de datos abstractos (TDA) (2)

- **Ejemplo: Stack (almacenamiento de datos en una pila)**
- **Típicas operaciones:**
  - push
  - pop
  - top
  - empty
  - new (creador)

Tipos de datos abstractos (TDA) (3)

Contratos: Forma de definir los derechos y deberes de los clientes y proveedores

- require: precondición que impone los deberes de el cliente
- postcondición: postcondición que impone los deberes del desarrollador de software
- invariante: expresa propiedades semánticas del TDA, y restricciones de integridad de la clase que lo implementa. Estas deben ser mantenidas por todas sus rutinas

Tipos de datos abstractos (TDA) (4)

Elementos de una especificación formal

- **tipos**
- **funciones: definen las operaciones (interfaz) del TDA**
- **axiomas: definen el comportamiento del TDA**
- **precondiciones: definen parte del contrato**

Tipos de datos abstractos (TDA) (5)

Ejemplo: Stack

- **Tipo**
  - STACK[G], donde G es un tipo arbitrario
- **Funciones**
  - push: STACK[G]xG → STACK[G]
  - pop: STACK[G] ↛ STACK[G]
  - top: STACK[G] ↛ G
  - empty: STACK[G] → boolean
  - new: → STACK[G]

Tipos de datos abstractos (TDA) (6)

¿Por qué funciones y no procedimientos?

- TDA es un modelo matemático
- Concepto de función ya existe

Clasificación de funciones:

- Creación (new)
- Consulta (top, empty). Obtención de propiedades
- Comando (pop, push). Obtención de nuevas instancias a partir de las ya existentes

Tipos de datos abstractos (TDA) (7)

- **Axiomas:**
  - top(push(s,x)) = x
  - pop(push(s,x)) = s
  - empty(new) = true
  - not empty(push(s,x)) = true

- **Precondiciones (funciones parciales)**
  - pop(s:STACK[G]) require not empty(s)
  - top(s:STACK[G]) require not empty(s)

De TDA's a clases

¿Qué es una clase? **TDA equipado con una implementación (variables de instancia y algoritmos), posiblemente parcial**

- **Clase efectiva:** **especifica completamente su implementación**
- **Clase abstracta:** **puede esta implementada parcialmente o nada**

¿Cuál es la forma de definir una clase efectiva?

- Especificar un TDA
- Escoger una implementación
- Implementar las funciones, axiomas y precondiciones

De TDA's a clases (2)

¿Cómo se implementa una clase efectiva?

- Parte pública: funciones del TDA
- Parte privada: parte dependiente de la implementación

El TDA se implementa como sigue:

- Comandos procedimientos
- Consultas funciones
- Axiomas postcondicones o invariantes
- Precondiciones precondiciones

De TDA's a clases (3)

Aspectos de C++ que permiten la implementación de un TDA

- **Parte pública y privada**
- **Inicialización y limpieza de objetos (constructor y destructor)**
- **Asignación e inicialización (redefinir operador de asignación y redefinir constructor de copia)**
- **Redefinición de operadores**
- **Llamada implícita y explícita de destructores**
- **Tipos parametrizados (templates)**

De TDA's a clases (4)

```
template<class T>
class stack{
  public:
    stack();
    stack(int);
    stack(stack<T>& s);
    ~stack();
    void push(T elemento);
    void pop();
    T top();
    int empty();
  private:
    T* contenedor;
    int tamano;
    int tope;

    int operator==(stack<T>& s);
    int invariante(){
        return tamano > 0 && tope >=-1 && tope < tamano && contenedor != NULL;
    }
};
```


De TDA's a clases (5)

```
template<class T> stack<T>::stack(){
    assert(0);
}
template<class T> stack<T>::stack(int t){
    assert(t>0);
    tamano = t;
    tope = -1;
    contenedor = new T[t];
    assert(invariante());
}
template<class T> stack<T>::stack(stack<T>& s){
    assert(s.invariante());
    tamano = s.tamano;
    tope = s.tope;
    contenedor = new T[s.tamano];
    for(int i=0; i<=tope; i++)
         contenedor[i] = s.contenedor[i];
    assert(invariante() && s.invariante() && s == *this);
}
```


De TDA's a clases (6)

```
template<class T> stack<T>::~stack(){
    assert(invariante());
    delete contenedor;
}
template<class T> void stack<T>::push(T elemento){
    assert(invariante() && (tope+1) < tamano);
    contenedor[tope+1] = elemento;
    tope +=1;
    assert(contenedor[tope] == elemento &&
           invariante());
}
template<class T> void stack<T>::pop(){
    assert( !empty() && invariante());
    tope -=1;
}
```


De TDA's a clases (7)

```
template<class T> T stack<T>::top(){
    assert( !empty() && invariante());
    return contenedor[tope];
}

template<class T> int stack<T>::operator==(stack<T>& s){
    assert( invariante() || s.invariante() );

    /* solo el contenido debe der igual */
    if( tope != s.tope ) return 0;

    int i;
    for(i=0; i <= tope;  i++ )
        if( contenedor[i] != s.contenedor[i]) return 0;
    return 1;
}
```


De TDA's a clases (8)

File: main.C

```
#include <iostream>
#include "stack.h"
main(){
    stack<int> s(10);
    s.push(5);
    s.push(8);
    s.push(3);
    s.push(-1);
    s.push(20);
    stack<int> i_stack=s; // inicializacion
    std::cout << "Contenido del stack:\n";
    while( !s.empty() ){
        std::cout << s.top() << "\n";
        s.pop();
    }
    std::cout << "Contenido del stack inicializado:\n";
    while( !i_stack.empty() ){
            std::cout << i_stack.top() << "\n";
            i_stack.pop();
    }
}
```


De TDA's a clases (9)

File: main.C

```
    stack<float> *f_stack;
    f_stack = new stack<float>(20);

    f_stack->push(5);

    delete f_stack; // destruccion explícita
    // destrucción implícita de s y i_stack
}
```


De TDA´s a clases

Aspectos importantes en Java

- **Parte pública y privada**
- **Inicialización (constructor)**
- **Definición de objetos genéricos a través del tipo Object**
- **Recolección automática de basura**

De TDA's a clases (10)

Stack generico en java: (version simple)

file: pila.java

```
class pila{

    private Object  v[];
    private int tope;
    private int max_size;
    public pila(int size){
        max_size = size;
        v = new Object[size];
        tope = -1;
    }
    public Object Tope(){
        return v[tope];
    }
    public void Pop(){ tope--;}
    public void Push(Object item){
        v[++tope] = item;
    }
    public boolean Empty(){
         return tope == -1;
    }
}
```


De TDA's a clases (11)

Stack genérico en java: (version simple)

file: Ejemplo.java

```
import Pila;

public class Ejemplo {

    public static void main(String args[]){

            Pila s1 = new Pila(100);

            s1.push(new Integer(1));
            s1.push(new Integer(2));
            s1.push(new Integer(3));

            while( !s1.Empty() ){
                 System.out.print(s1.tope() + " ");
                 s1.pop();
            }
    }
}
```


De TDA´s a clases

```
class Pila{

    private Object  v[];
    private int tope;
    private int max_size;

    private bolean invariante(){ return -1 <= tope && tope < max_size && v !=null;}
    public Pila(int size){
        assert size > 0 : "tamano inalido";
        max_size = size;
        v = new Object[size];
        tope = -1;
        assert invariante() : "invariante invalido";
    }
    public Object top(){
        assert invariante() : "invariante invalido";
        assert !empty() : "No hay elementos en la pila";
        return v[tope];
    }
    public void pop(){
        assert invariante() : "invariante invalido";
        assert !empty() : "No hay elementos en la pila";
        tope--;
       assert invariante() : "invariante invalido";
    }
```


De TDA's a clases

```
 public void Push(Object item){
        assert (tope < max_size+1) : "pila llena";
        assert invariante() : "invariante invalido";
        v[++tope] = item;
        assert invariante() : "invariante invalido";
    }
    public boolean Empty(){
        assert invariante() : "invariante invalido";
        return tope == -1;
    }
}
```