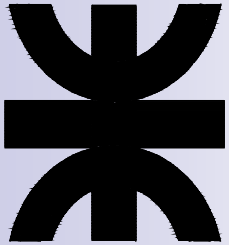


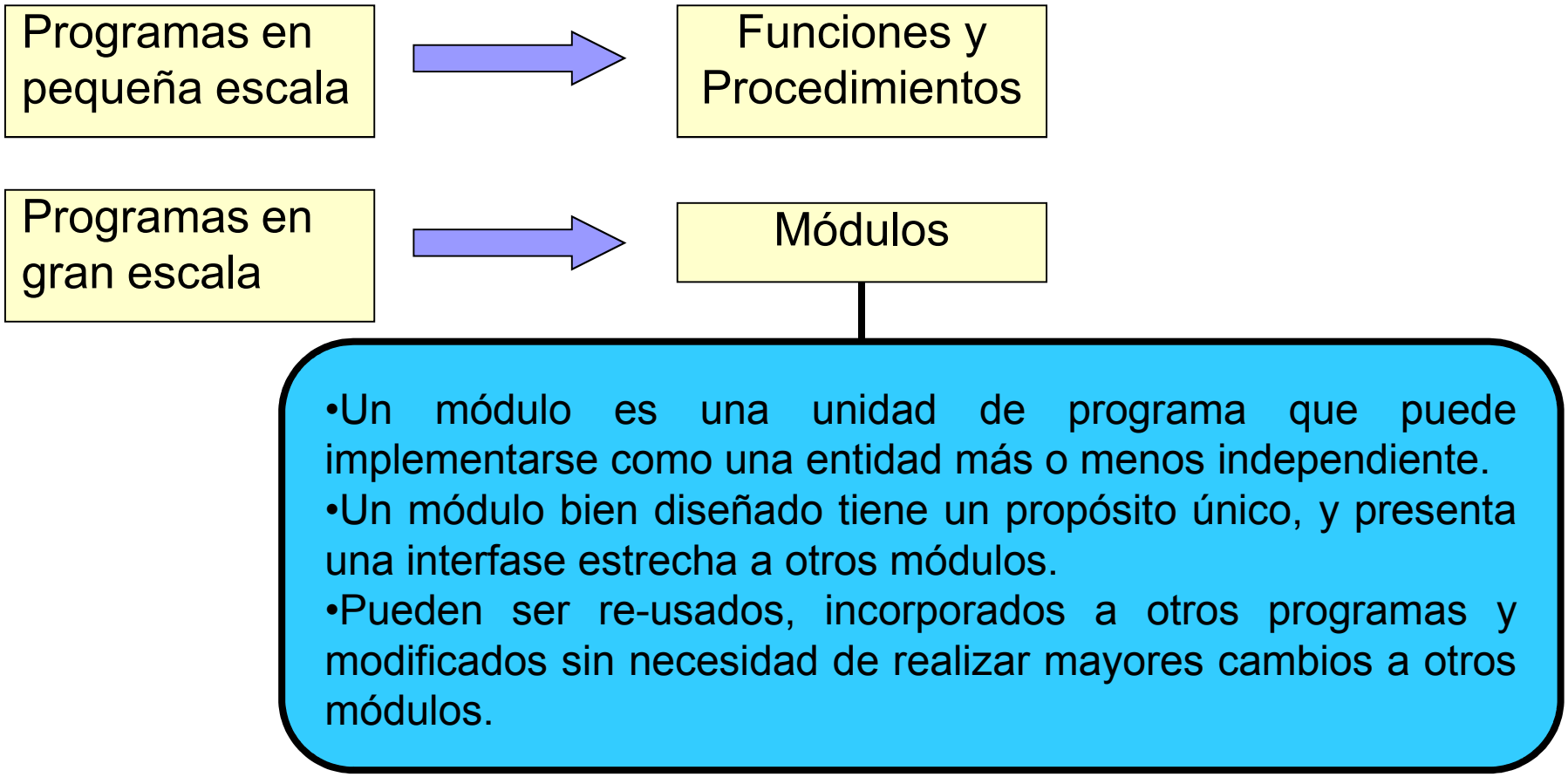
Universidad Tecnológica Nacional
Facultad Regional Resistencia

Paradigmas de Programación
Programación Orientada a Objetos

Programación en Grande
Complejidad

- **Tamaño del software**
  - Hace dos/tres décadas: programas en lenguaje ensamblador en torno a centenares de líneas.
  - Hoy: lenguajes de alto nivel con centenares de millares, o incluso millones de líneas de código.
- **Ámbitos de la Complejidad**
  - *Complejidad del problema*: la implementación se descompone en centenares y a veces miles de módulos independientes que implica tener un equipo de desarrolladores.
  - *Complejidad "humana"*: cuantos más desarrolladores, las comunicaciones entre ellos son más complejas, la coordinación es difícil: equipos dispersos geográficamente (proyectos grandes)
- **Propuestas de reutilización (reusability) de componentes software:**
  - Bloques iniciales para la construcción del programa, similares a la construcción de cualquier objeto complejo (tal como un automóvil) que se construye ensamblando sus partes.

Modularidad

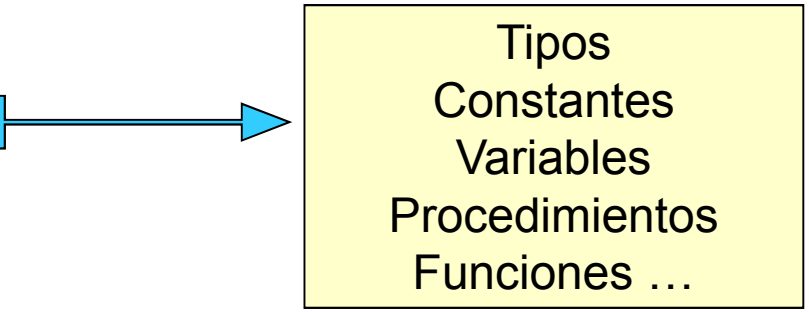


Módulos

- Son importantes dos cuestiones respecto de los módulos:
  - *Que* es lo que hace el módulo?
    - *Su propósito.*
    - *Cuál es el uso que le podemos dar.*
    - *Le interesa al usuario del módulo.*
  - *Cómo* se consigue ese propósito?
    - *Le interesa solo al implementador.*

Módulos

- Un módulo es un grupo de varios *componentes* declarados con un propósito común.



Existen varios conceptos importantes que soportan la modularidad: paquetes (*package*), tipos abstractos de datos, objetos, clases de objetos y genéricos (*generics*).

Modularidad
Paquetes

- Es un grupo de componentes declarados.
- En general, los componentes declarados pueden ser tipos, constantes, variables, procedimientos, funciones y sub-paquetes.

```
Package seguridad is
  CantMinCaract: constant integer := 6;
            -- Cantidad Mínima de Caracteres en las Contraseñas
  PerCambioClave: constant integer := 15;
            -- Periodo de cambio de claves.
  CantClavesHist: constant integer := 5;
            -- Cantidad de Claves en historial.
end seguridad;
```

- Esto produce una asociación de o un enlace entre `seguridad` y un conjunto de enlaces encapsulados.
  `{CantMinCaract → 6, PerCambioClave → 15, CantClavesHist → 5 }`

- Se pueden acceder mediante `seguridad.CantMinCaract`

Modularidad
Tipo Abstracto de Datos (TAD)

- **Tipo**: agrupación de elementos con características similares.
- **Abstracto**: no concreto, conceptual.
- **Dato**: información que una computadora puede entender.

- *Un Tipo Abstracto de Datos es un conjunto de valores cuya creación y conocimiento de sus características solo puede realizarse mediante un conjunto de operaciones.*
- Es un tipo cuya representación como tipo concreto ha sido abstraída y cuyos datos solo se pueden acceder a través de un conjunto de operaciones.
- El grupo de operaciones usualmente son constantes, funciones y procedimientos.
- El programador elige la representación de los valores del tipo abstracto, e implementa las operaciones en términos de esa representación elegida.

TAD
Ejemplo

Tipo Abstracto: racional

```
abstype racional = rac of (int * int)
with
    val cero = rac (0, 1)
    and uno  = rac (1, 1);

    fun op // (m: int, n: int) =
          if n <> 0
          then rac (m, n)
          else ... (* número racional inválido *)

    and op ++ (rac (n1, d1) : racional, rac (n2, d2) racional) =
          rac (n1*d2 + n2*d1, d1*d2)

    and op == (rac (n1, d1) : racional, rac (n2, d2) racional) =
          (n1*d2 = n2*d1)
end
```

TADs “Pedefinidos”

- Cualquier tipo predefinido del lenguaje puede considerarse un TAD: conjunto de valores + operaciones para su manejo.
- De acuerdo con las definiciones anteriores los tipos del lenguaje pueden considerarse *tipos abstractos de datos predefinidos*.

```
package Booleanos is
     type Booleano is private;
     function "y"  (A, B : Booleano) return Booleano;
     function "o"  (A, B : Booleano) return Booleano;
     function "no" (A : Booleano) return Booleano;
     Verdadero : constant Booleano;
     Falso : constant Booleano;
private
     type Booleano is (V,F);
     Verdadero : constant Booleano := V;
     Falso : constant Booleano := F;
end Booleanos;
```

Encapsulamiento
Ocultamiento de Información

- Comúnmente los paquetes contienen declaración de componentes exportados y ocultos, estos últimos sirven solo para dar soporte a la implementación de los componentes exportados.
- Los mecanismos de ocultación de los lenguajes hacen que los programas clientes conozcan sólo la interfaz (los servicios). De esta forma, se abstraen los detalles de implementación.
- Los lenguajes de programación ofrecen mecanismos para ocultar e impedir a los clientes el uso de ciertos elementos de nuestro código.
- Los módulos tienen dos partes bien diferenciadas
  - Una parte pública o interfaz.
  - Una parte privada o implementación.

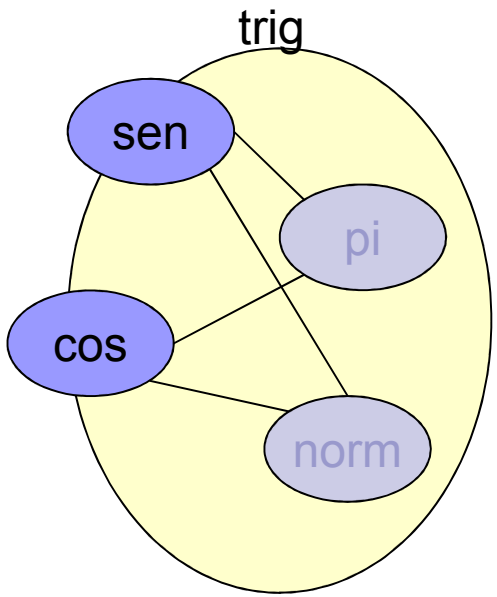
Ocultamiento de Información
Ventajas

- Facilita la tarea del programador: abstracción de los detalles.
- Facilita la reusabilidad del código.
- Permite el desarrollo independiente y paralelo de aplicaciones.
- Permite el desarrollo de versiones mejoradas de bibliotecas sin afectar al código que las utiliza.
- Permite distribuir implementaciones empaquetadas.

Ocultamiento de Información
Ejemplo: Paquetes

```
Package trig is
    function sen (x: float) return float;
    function cos (x: float) return float;
End trig;

Package body trig is
    Pi : constant float := 3.1416;
    Function norm (x: float) return float is
    ... ; -- retorna x modulo 2*pi
    Function sen (x: float) return float is
    ... ; -- retorna el seno de norm(x)
    Function cos (x: float) return float is
    ... ; -- retorna el coseno de norm(x)
end trig;
```



Ocultamiento de Información
Clases y Objetos

- Los objetos son otro tipo de módulo muy especial e importante, que consisten de datos ocultos junto con un conjunto de operaciones exportadas.
- Los mismos tendrán un tiempo de vida debido a que es un componente variable y se comporta como una variable común.
- Los Objetos se agrupan en clases de objetos similares.

Modulos y Ocultamiento

Ada: Generic Package

```
generic package directorio is
    procedure insertar (NombreNuevo : in Name; NumeroNuevo : in Number);
    procedure buscar   (NombreBusc  : in Name; NumeroBusc  : out Number;
                        Encontrado  : out boolean);
end clase_directorio;

package body clase_directorio is
type Nodo;
type Punt is access Nodo;
type Nodo is record
               Nombre : Name;
               Numero : Number;
               Der, izq : Punt;
             End record;
Raiz : punt;

procedure insertar (NombreNuevo : in Name; NumeroNuevo : in Number) is ... ;
    -- Implementación
procedure buscar   (NombreBusc  : in Name; NumeroBusc  : out Number;
                    Encontrado  : out boolean) is ... ;
    -- Implementación
end Clase_directorio;
```

Componentes Exportados

Componentes Ocultos

UTN
FRRe

Modulos y Ocultamiento
Ada: Generic Package

Para instanciar objetos de la clase directorio:

```
package dir_casa is new directorio;
package dir_trabajo is new directorio;
```

Para acceder a los componentes exportados utilizamos la notación punto:

```
dir_casa.insertar (yo, 452546);
dir_trabajo.insertar (yo, 479515);
dir_trabajo.buscar (yo, miNumero, Ok);
```

Modulos y Ocultamiento
C++: Class

```
class cola {
  int c[100];
  int posIni, posFin;
public
  cola (void);    // constructor
  void ponc (int i);
  int quitac (void);
};

cola::cola (void)
  {  posIni = posFin = 0;  }

void cola::ponc (int i)
  {  // implementación de la función }

int cola::quitac (void)
  {  // implementación de la función }
```

Componentes Ocultos

Componentes Exportados

Para instanciar un objeto de la clase cola:

```
cola colaEnteros1;
cola colaEnteros2;
```

Clases y TADs
Similitudes y Diferencias

- Ambos conceptos permiten crear variables del mismo tipo cuya representación es oculta.
- En ambos casos los accesos a esas variables solo puede hacerse a través de operaciones que se proveen para ese propósito.

Clases y TADs
Similitudes y Diferencias

- Los TAD carecen de:
  - Extensibilidad de tipos (Herencia)
  - Inicialización automática de objetos (Constructores).
  - Finalización automática de objetos (Destructores).
  - Selección de Operaciones en tiempo de ejecución.

Clases y TADs
Similitudes y Diferencias

Diferencias

TAD

```
procedure insertar (dir : in out Directorio;
                    NombreNuevo : in Name; NumeroNuevo : in Number);
procedure buscar   (dir : in Directorio;
                    NombreBusc : in Name; NumeroBusc  : out Number;
                    Encontrado : out boolean);
```

```
insertar (dir_casa, 'María', 412514)
```

Clase

```
procedure insertar (NombreNuevo : in Name; NumeroNuevo : in Number);

procedure buscar   (NombreBusc  : in Name; NombreBusc  : out Number;
                    Encontrado  : out boolean);
```

```
dir_casa.insertar ('María', 412514)
```