Metodología de la Programación

Conv. Ordinaria 2024/25 — Tienda de Orlas

Repaso completo con solución — Para revisar en iPad

Estructura de clases

```
enum class TipoPersona {Estudiante, Profesor};

class Cliente {              // Simple, sin punteros     NO necesita Big Four
    string DNI, nombreApellidos, ciudad;
    TipoPersona rol;
    // DADO: constructores, set/get, operator== y !=, operator<< y >>
};

class Orla {                 // Array dinamico de Cliente     Big Four DADO
    int idOrla;
    string grado, curso;
    Cliente* clientes;       int capacidad;    int registrados;
    // DADO: Big Four, getCapacidad, getRegistrados, estaAbierta, idOrla/grado/curso
    // DADO: operator[], bool contiene(string& unDNI), operator<< y >>
    // DADO: getNumEstudiantes(), getNumProfesores()
};

class Tienda {               // Array dinamico de Orla     Big Four: LO
    IMPLEMENTAMOS
    string nombre;
    Orla* orlas;             int capacidad;    int usados;
    // DADO: setNombre, getNombre, getUsados, operator[]
    // IMPORTANTE: primero van orlas ABIERTAS, luego CERRADAS
};
```

Ejercicio 1 — Métodos para la clase `Tienda` [1.5 puntos]

1. Constructor para crear un objeto `Tienda` vacía (con cero orlas), con un parámetro entero para establecer la *capacidad* de la tienda y que valdrá cero por defecto.
2. Constructor de copia.
3. Destructor.
4. Operador de asignación.

```
// --- Metodos PRIVADOS (allocate / deallocate / copy) ---

void allocate(int cap) {
    capacidad = (cap > 0) ? cap : 0;
    orlas     = (capacidad > 0) ? new Orla[capacidad] : nullptr;
}

void deallocate() {
    delete[] orlas;
    orlas     = nullptr;
    capacidad = 0;
    usados    = 0;
}

void copy(const Tienda& otra) {
    nombre = otra.nombre;          // copiar TODOS los datos miembro
    usados = otra.usados;
    for (int i = 0; i < usados; i++)
        orlas[i] = otra.orlas[i];
}
```

```
21
22  // --- Constructor con parametro (cubre tambien el caso sin argumentos) ---
23  Tienda(int cap = 0) {
24      nombre = "";
25      usados = 0;
26      allocate(cap);
27  }
28
29  // --- Constructor de copia ---
30  Tienda(const Tienda& otra) {
31      allocate(otra.capacidad);
32      copy(otra);
33  }
34
35  // --- Destructor ---
36  ~Tienda() { deallocate(); }
37
38  // --- Operador de asignacion ---
39  Tienda& operator=(const Tienda& otra) {
40      if (this != &otra) {          // autocheck OBLIGATORIO
41          deallocate();
42          allocate(otra.capacidad);
43          copy(otra);
44      }
45      return *this;
46  }
```

Error tipico: no copiar `nombre` en `copy()` — el operador= entonces no actualiza el nombre. **Otro error tipico:** poner `allocate(0)` en lugar de `allocate(cap)` en el constructor.

Ejercicio 2 — Sobrecarga de operadores [2.5 puntos]

1. **operator+= para `Orla`**: añade un `Cliente` al final del array `clientes`. Si la orla está cerrada o el cliente ya está, no hace nada.
2. **operator-= para `Orla`**: recibe un DNI y elimina el cliente con ese DNI desplazando hacia la izquierda. Si no existe, no hace nada.
3. **operator+= para `Tienda`**: añade una `Orla`. Si está *cerrada* → al **final**. Si está *abierta* → al **inicio** (desplaza el resto a la derecha). Si no hay espacio, redimensiona **duplicando**. Si la capacidad es cero, empieza con 1.

```
1   // --- operator+= Orla (aniade Cliente) ---
2   Orla& operator+=(const Cliente& cliente) {
3       if (!estaAbierta() || contiene(cliente))  // cerrada O ya existe: no hacer nada
4           return *this;
5       clientes[registrados] = cliente;
6       registrados++;
7       return *this;
8   }
9
10  // --- operator-= Orla (elimina por DNI) ---
11  Orla& operator-=(const string& DNIbuscado) {
12      for (int i = 0; i < registrados; i++) {
13          if (clientes[i].getDNI() == DNIbuscado) {
14              for (int j = i; j < registrados - 1; j++)
15                  clientes[j] = clientes[j+1];
16              registrados--;
17              return *this;          // sale en cuanto lo encuentra
18          }
19      }
20      return *this;
21  }
```

```
22
23 // --- operator+= Tienda (aniade Orla al inicio o al final) ---
24 Tienda& operator+=(const Orla& orla) {
25     if (usados == capacidad) {
26         int newCap = (capacidad == 0) ? 1 : capacidad * 2;
27         Orla* nuevo = new Orla[newCap];
28         for (int i = 0; i < usados; i++)
29             nuevo[i] = orlas[i];
30         delete[] orlas;
31         orlas     = nuevo;
32         capacidad = newCap;
33     }
34     if (!orla.estaAbierta()) {
35         orlas[usados] = orla;          // cerrada     al final
36     } else {
37         for (int i = usados; i > 0; i--)
38             orlas[i] = orlas[i-1];     // abierta     desplazar derecha
39         orlas[0] = orla;               // poner al inicio
40     }
41     usados++;
42     return *this;
43 }
```

Diferencia con otros examenes: aqui el `+=` de Tienda inserta al inicio o al final segun si la orla esta abierta o cerrada. La logica **duplica** (no suma TAM_BLOQUE). El `||` en `operator+=` de Orla es critico — con `&&` seria un bug grave.

Ejercicio 3 — Métodos y funciones para E/S [2 puntos]

1. Sobrecarga los operadores « y » para la clase `Tienda`. Recuerda que tienes disponibles « y » para las clases `Orla` y `Cliente`. El formato es:

   ```
   TIENDA: ORLAS DE INGENIERIA
   ORLAS: 3
   ID_ORLA: 223
   2024-2025 Ingenieria Informatica
   Clientes: 64  Registrados: 3
   ...Cliente1...   ...Cliente2...   ...Cliente3...
   ID_ORLA: 256
   ...
   ```

2. Implementa un constructor de `Tienda` con un parámetro con el nombre de un fichero. La primera línea contendrá la palabra mágica `TIENDA_ORLAS`. Si no coincide, crear una tienda vacía llamada `DATOS INCORRECTOS`.

```
1  // --- operator<< ---
2  friend ostream& operator<<(ostream& flujo, const Tienda& tienda) {
3      flujo << "TIENDA:␣" << tienda.nombre << endl;
4      flujo << "ORLAS:␣"  << tienda.usados << endl;
5      for (int i = 0; i < tienda.usados; i++)
6          flujo << tienda.orlas[i];     // << de Orla ya lleva su propio endl
7      return flujo;
8  }
9
10 // --- operator>> ---
11 friend istream& operator>>(istream& flujo, Tienda& tienda) {
12     string etiqueta;
13     flujo >> etiqueta;                // descarta "TIENDA:"
14     flujo.ignore();                   // descarta el espacio
15     getline(flujo, tienda.nombre);    // lee nombre con posibles espacios
16
```

```
    int nOrlas;
    flujo >> etiqueta >> nOrlas;      // descarta "ORLAS:" y lee numero

    for (int i = 0; i < nOrlas; i++) {
        Orla orla;
        flujo >> orla;                // usa >> de Orla (disponible)
        tienda += orla;               // usa += de Tienda (ej2)
    }
    return flujo;
}

// --- Constructor desde fichero ---
Tienda(const string& nombreFichero) {
    nombre = "";
    usados = 0;
    allocate(0);

    ifstream fichero(nombreFichero);
    if (fichero.is_open()) {
        string magic;
        getline(fichero, magic);
        if (magic == "TIENDA_ORLAS") {
            fichero >> *this;         // reutiliza operator>>
        } else {
            nombre = "DATOS␣INCORRECTOS";   // nombre = no ==
        }
        fichero.close();
    }
}
```

Bug frecuente: escribir `nombre == "DATOS INCORRECTOS"` (comparacion) en lugar de `nombre = "DATOS INCORRECTOS"` (asignacion). Compila sin error pero no hace nada.

Ejercicio 4 — Otros métodos o funciones de cálculo [2 puntos]

En la clase `Tienda` implementa:

1. Métodos `numOrlasAbiertas` y `numOrlasCerradas`.
2. Operadores relacionales `==` y `>` entre dos `Tienda`. `tiendaA >tiendaB` si el porcentaje de orlas cerradas de A es mayor que el de B.
3. Método `aniadeCliente` que recibe un `Cliente` y un identificador de orla. Añade el cliente a esa orla. Si al añadirlo la orla se cierra, muévela al final (desplaza izquierda). Si no se encuentra la orla, no hace nada.

```
// --- 4.1: contar orlas abiertas y cerradas ---
int numOrlasAbiertas() const {
    int contador = 0;
    for (int i = 0; i < usados; i++)
        if (orlas[i].estaAbierta()) contador++;
    return contador;
}

int numOrlasCerradas() const {
    return usados - numOrlasAbiertas();  // reutiliza el anterior
}

// --- Helper PRIVADO para los operadores (4.2) ---
float porcentajeCerradas() const {
    if (usados == 0) return 0.0f;
    return (float)numOrlasCerradas() / usados;
}
```

```
18
19 // --- 4.2: Operadores relacionales (como friend dentro de la clase) ---
20 friend bool operator>(const Tienda& a, const Tienda& b) {
21     return a.porcentajeCerradas() > b.porcentajeCerradas();
22 }
23 friend bool operator==(const Tienda& a, const Tienda& b) {
24     return a.porcentajeCerradas() == b.porcentajeCerradas();
25 }
26
27 // --- 4.3: aniadeCliente ---
28 void aniadeCliente(const Cliente& cliente, int idOrla) {
29     // 1. Buscar la orla por id
30     int pos = -1;
31     for (int i = 0; i < usados && pos == -1; i++)
32         if (orlas[i].getIdOrla() == idOrla)
33             pos = i;
34
35     if (pos == -1) return;         // no encontrada: no hacer nada
36
37     // 2. A adir el cliente (usa += de Orla)
38     orlas[pos] += cliente;
39
40     // 3. Si se cerro al a adirlo: mover al final
41     if (!orlas[pos].estaAbierta()) {
42         Orla laCerrada = orlas[pos];
43         for (int i = pos; i < usados - 1; i++)
44             orlas[i] = orlas[i+1];         // desplazar izquierda
45         orlas[usados-1] = laCerrada;       // poner al final
46         // usados NO cambia: mismas orlas, reordenadas
47     }
48 }
```

Patron identico en todos los examenes: metrica privada → operadores relacionales → mover elemento. El orden importa: añadir PRIMERO, comprobar si se cerro DESPUES.

Ejercicio 5 — Aplicación: Cadena de Tiendas [2 puntos]

La empresa "ORLAS SALRO" gestiona un conjunto de tiendas. Implementa un programa que reciba de la línea de órdenes un número indeterminado de ficheros de datos `Tienda` y que:

1. Construya un *array dinámico* de tiendas cargadas desde los ficheros.
2. Calcule la "mejor" (mayor porcentaje de orlas cerradas) y la "peor" y las muestre por la salida estándar.
3. Para cada tienda, calcule y muestre el porcentaje promedio de clientes sin registrar en las orlas abiertas.

```
1 // --- Helper en Tienda: promedio de huecos libres en orlas abiertas ---
2 float mediaSinRegistrar() const {
3     float suma = 0.0;
4     for (int i = 0; i < usados; i++)
5         if (orlas[i].estaAbierta())
6             suma += (float)(orlas[i].getCapacidad() - orlas[i].getRegistrados())
7                     / orlas[i].getCapacidad();
8     int abiertas = numOrlasAbiertas();
9     return (abiertas > 0) ? suma / abiertas : 0.0f;
10 }
11
12 // --- main ---
13 int main(int argc, char* argv[]) {
14     int n = argc - 1;
15
```

```
    // 1. Array dinamico de tiendas
    Tienda* tiendas = new Tienda[n];
    for (int i = 0; i < n; i++)
        tiendas[i] = Tienda(argv[i + 1]);   // constructor desde fichero

    // 2. Mejor y peor (usando operator> ya implementado)
    int posMejor = 0, posPeor = 0;
    for (int i = 1; i < n; i++) {
        if (tiendas[i] > tiendas[posMejor]) posMejor = i;
        if (tiendas[posPeor] > tiendas[i])  posPeor  = i;
    }
    cout << "Mejor tienda:" << endl << tiendas[posMejor];
    cout << "Peor tienda:"  << endl << tiendas[posPeor];

    // 3. Porcentaje sin registrar por tienda
    for (int i = 0; i < n; i++)
        cout << tiendas[i].getNombre() << ": "
             << tiendas[i].mediaSinRegistrar() << "%" << endl;

    delete[] tiendas;
    return 0;
}
```

Peor = menor porcentaje de cerradas = `tiendas[posPeor] >tiendas[i]` actualiza el peor cuando encontramos una tienda con MENOS cerradas que la actual peor.

Ejercicio	Puntos	Lo mas importante
Ej1 — Big Four Tienda	1.5	copy() debe incluir nombre; allocate(cap) no allocate(0)
Ej2 — operadores +=/-=	2.5	—— no && en Orla; duplicar en Tienda; caso cap==0
Ej3 — E/S + constructor file	2.0	ignore()+getline; cadena magica; nombre= no nombre==
Ej4 — Calculo	2.0	porcentajeCerradas() privado; añadir antes de comprobar
Ej5 — Main	2.0	argc-1 ficheros; delete[]; mediaSinRegistrar solo abiertas