Guía de Ejercicios 6: Introducción a Tipos Abstractos de Datos

[Version: 27 de octubre de 2024]

Objetivos:

- Repasar los conceptos de clase, método y representación interna.
- Introducir el concepto de invariante de representación y entrenar su lectura y su escritura para estructuras simples.
- Entrenar la implementación de métodos de una clase preservando el invariante de representación.

Ejercicio 1. Se desea implementar una clase `Libro` que lleva registro de cuánto hemos leídos de un determinado libro.

Interfaz:

- `Libro(string titulo, string autor, int cantidad_paginas)`: construye e inicializa un nuevo objeto de la clase `Libro`.
- `lib.paginas_totales()`: indica la cantidad de páginas totales que tiene el libro `lib`.
- `lib.pagina_actual()`: indica el número de página a la que llegamos hasta el momento.
- `lib.porcentaje_leido()`: indica el porcentaje del total del libro `lib` que ya leímos (contando como que ya leímos desde el principio hasta la página actual).
- `lib.avanzar_paginas(int n)`: avanza la página actual hasta `n` páginas más adelante.
- `lib.saltar_a_pagina(int k)`: salta la página actual hasta la página número `k`.
- `lib.finalizado()`: indica si ya terminamos de leer el libro completo o no.

Estructura de representación:

```
1  string titulo;
2  string autor;
3  int paginas_totales;
4  int pagina_actual;
```

(a) ¿Cuál es el invariante de representación $Rep(e : estr)$ adecuado para la estructura?

(I) $Rep(e : estr) \equiv e.paginas\_totales > 0$

(II) $Rep(e : estr) \equiv |e.titulo| > 0 \ \wedge \ |e.autor| > 0 \ \wedge \ e.paginas\_totales > 0 \wedge 0 < e.pagina\_actual \leq e.paginas\_totales$

(III) $Rep(e : estr) \equiv |e.titulo| > |e.autor| \ \wedge \ e.paginas\_totales > 0 \wedge 0 < e.pagina\_actual \leq e.paginas\_totales$

(b) Tomando el invariante de representación elegido, dar un ejemplo que lo cumpla y uno que no. Es decir, dar dos instancias e_1 y e_2 tal que $Rep(e_1)$ es verdadero pero $Rep(e_2)$ es falso.

(c) ¿Es necesario agregar alguna precondición a los métodos de la clase? ¿Cuáles?

(d) Implementar la clase `Libro` en C++ respetando el invariante de representación elegido.

Ejercicio 2. Se quiere una clase `Tateti`. Debe proveer métodos para crear un juego, determinar de quién es el turno, hacer una jugada (cruz o círculo), determinar si el juego terminó, y (cuando el juego terminó) determinar quién ganó o si hubo empate. Por simplicidad, **suponer que siempre empieza jugando la cruz**.

```
enum class Casillero {circulo, cruz, vacio};
enum class Jugador {circulo, cruz};

class Tateti{
    vector<vector<Casillero> > tablero;

  public:
    Tateti();
    Jugador a_quien_le_toca();
    Casillero observar_posicion(int fila, int columna);
    void hacer_jugada(Jugador, int fila, int columna);
    bool esta_terminado();
    bool hubo_empate();
    Jugador quien_gano();
};
```

(a) ¿Cuáles de las siguientes condiciones deberían estar incluídas en el invariante de representación y cuales no?

(I) La cantidad de `cruz` y `circulo` en `tablero` es la misma o la cantidad de `cruz` es la cantidad de `circulo` más uno.

(II) Cada elemento de `tablero` es `cruz`, `circulo` o `vacio`.

(III) El tamaño de `tablero` es 3x3.

(IV) Si hay 3 elementos iguales (no `vacio`) en una fila, columna o diagonal, entonces el juego terminó.

(V) Si en `tablero` no hay ningún `vacio`, entonces el juego terminó.

(b) ¿Cuál es la precondición del método `quien_gano`? ¿Qué otros métodos tienen Pre?

(c) Implementar la clase `Tateti` en C++.

Ejercicio 3. Para cada una de las siguientes clases

- Indicar qué métodos son observadores, constructores, modificadores y otras operaciones.
- Describir en español qué información devuelve cada observador.
- Escribir la pre y post condición de los constructores, modificadores y otras operaciones (puede ser en español).
- Escribir en español el invariante de representación de la estructura interna, mostrar un ejemplo de instancia que cumpla el invariante y uno que no lo cumpla.
- Implementar los métodos en C++ respetando el invariante propuesto.

(a)
```
class Fecha{
  public:
    Fecha(int dia, int mes, int anio);
    void avanzar_dia();
```

```
    void avanzar_n_dias(int n);
    int dia() const;
    int mes() const;
    int anio() const;
    bool operator==(const Fecha& f) const;

  private:
    int dia;
    int mes;
    int anio;
}
```

(b)
```
class Usuario{

  public:
    Usuario(string nombre, int edad);
    string nombre() const;
    int edad() const;
    void agregar_amigo(string nombre);
    bool es_amigo(string nombre) const;
    bool es_popular() const;
    // Un usuario se considera popular si tiene más de 10 amigos.

  private:
    string nombre;
    int edad;
    set<string> amigos;
    bool es_popular;
}
```

(c)
```
class Multiconjunto{
  public:
    Multiconjunto();
    void agregar(int e);
    void quitar(int e);
    int contar_apariciones(int e) const;
    int cardinal() const;
    int cantidad_elementos_distintos() const;

  private:
    int cantidad_distintos;
    vector<int> elementos;

  }
```

Ejercicio 4. Dada la siguiente interfaz y la estructura de representación elegida.

```
class Carrito{
  public:
    // Constructor
    Carrito(int peso_maximo);

    // Observadores
    int peso_disponible() const;
    bool buscar_item(string nombre) const;
```

```
    int peso_de_item(string nombre) const;
    int precio_de_item(string nombre) const;

    // Modificadores
    void borrar_item(string nombre);
    void agregar_item(string nombre, int peso, float precio);

    // Otras operaciones
    float monto_total() const;

  private:
    int peso_maximo;
    vector<string> items;
    map<string, int> pesos_items;
    map<string, float> precios_por_item;
};
```

(a) Los siguientes predicados deben ser parte del invariante $Rep(e : estr)$, describir en español qué significa cada uno.

(I) $e.peso\_maximo \geq 0$

(II) $sinRepetidos(e.items)$, donde $sinRepetidos$ es un pred. auxiliar previamente definido.

(III) $claves(e.pesos\_items) = claves(e.precios\_por\_item)$, donde $claves(m)$ devuelve el conjunto de claves del map m.

(IV) $(\forall s : \texttt{string})(esta?(s, e.items) \implies s \in claves(e.pesos\_items))$,
donde $esta?(s : \texttt{string}, v : \texttt{vector<string>})$ es un pred. auxiliar prev. definido.

(V) $(\forall s : \texttt{string})(s \in claves(e.pesos\_items) \implies esta?(s, e.items))$

(VI) $(\forall s : \texttt{string})(s \in claves(e.pesos\_items) \implies e.pesos\_items[s] > 0)$

(VII) $(\forall s : \texttt{string})(s \in claves(e.precios\_por\_item) \implies e.precios\_por\_item[s] > 0)$

(VIII) $\left(\sum_{s \in claves(e.pesos\_items)} e.pesos\_items[s]\right) \leq e.peso\_maximo$

(b) Los siguientes enunciados **no deben** ser parte del invariante $Rep(e : estr)$, para cada uno explicar por qué no.

(I) $e.peso\_maximo$ es de tipo `int`.

(II) $|e.items| \neq 0$.

(III) $e.pesos\_items$ es un `map` que tiene como claves los nombres de los items y como valores los pesos de los mismos.

(c) ¿Qué métodos de la clase `Carrito` necesitan una precondición?

Ejercicio 5. A continuación presentamos distintas estructuras de representación. Cada una es una estructura de representación alternativa para alguna de las clases de los ejercicios anteriores.

- Indicar a qué clase de los ejercicios anteriores corresponde.
- Escribir en español el invariante de representación de la nueva estructura.
- Identificar cuáles son los métodos de la clase correspondiente cuya implementación se debe modificar y cuáles no. Explicar cómo se modificarían.
- ¿Es necesario modificar las pre y post condiciones de los métodos? ¿Por qué?

(a)
```
private:
    string nombre;
    int edad;
    set<string> amigos;
    int cantidad_amigos;
```

(b)
```
private:
    int dia_del_anio;
    int anio;
```

(c)
```
  private:
    int peso_maximo;
    int peso_total;
    map<string, int> pesos_items;
    map<string, float> precios_por_item;
```

(d)
```
private:
    map<int,int> elementos;
```

Ejercicio 6. A continuación presentamos fragmentos de Clases en C++. Damos una estructura de representación junto a su invariante y también la implementación de algún método.

- Identificar si la implementación del método preserva el invariante o no.
- En caso de que no lo preserve, corregir la implementación del método.

(a)
```
class Conjunto {
public:
    /* ... */
    void agregar(int elem);
    /* ... */
private:
    vector<int> elementos;
};
```

$Rep(e : estr) \equiv sinRepetidos(e.elementos)$

```
void Conjunto::agregar(int e) {
    elementos.push_back(e);
}
```

(b)
```
class Reservas{
public:
    /* ... */

    void cancelar_reserva(string nombre);

    /* ... */
private:
    int cantidad_premium;
```

```
    set<string> listado;
    map<string, bool> es_premium;
};
```

$Rep(e : estr) \equiv$

- El conjunto de claves de $e.es\_premium$ es igual al conjunto $e.listado$.
- $e.cantidad\_premium$ es igual a la cantidad de claves de $e.es\_premium$ que están asociadas al valor `true`.

```
void cancelar_reserva(string nombre){
    listado.erase(nombre);
    es_premium.erase(nombre);
}
```

(c)
```
class Examenes{
public:
    /* ... */

    void agregar_examen(string nombre_alumno, int nota);
    // PRE: 0 <= nota <= 10 y nombre_alumno no fue ingresado
    // al sistema previamente.

    /* ... */
private:
    set<string> alumnos_aprobados;
    map<string, int> notas_por_alumno;
};
```

$Rep(e : estr) \equiv$ el conjunto $e.alumnos\_aprobados$ son todas las claves del diccionario $e.notas\_por\_alumno$ que están asociadas a un valor mayor o igual a 4.

```
void agregar_examen(string nombre_alumno, int nota){
    notas_por_alumno[nombre_alumno] = nota;
}
```

(d)
```
class Inmobiliaria{
public:
    /* ... */

    set<string> casas() const;
    bool esta_disponible() const;

    void cambiar_a_no_disponible(string direccion);
    // PRE: la casa 'direccion' ya se encuentra en el
    // sistema y está disponible.

    /* ... */
private:
    set<string> casas_disponibles;
    set<string> casas_no_disponibles;
    map<string, bool> casas_todas;
};
```

$Rep(e : estr) \equiv$

- El conjunto $e.casas\_disponibles$ son todas las claves del diccionario $e.casas\_todas$ que están asociadas al valor `true`.
- El conjunto $e.casas\_no\_disponibles$ son todas las claves del diccionario $e.casas\_todas$ que están asociadas al valor `false`.

```
void cambiar_a_no_disponible(string direccion){
    casas_todas[direccion] = false;
}
```

Ejercicio 7. Como usuario de los tipos abstractos de datos presentados en **??** y **??**, escribir el código que realice las siguientes acciones:

(a) Construir una `Fecha` con tu fecha de nacimiento y otra `Fecha` con la fecha de tu primer cumpleaños, avanzar tu fecha de nacimiento `365` días y comprobar si ambas fechas son iguales.

(b) Construir un `Usuario` con tu nombre y edad, agregar al menos cinco amigos, consultar si una persona está o no en entre tus amigos.

(c) Construir el siguiente `Multiconjunto`: $\{2, 65, 2, 3, 8, 8, 7, 65, 0\}$ y consultar la cantidad de elementos distintos.

(d) Construir un `Carrito` de peso máximo 10 con los items necesarios para cocinar lo que cenaste ayer, consultar el peso disponible y el monto total.

Ejercicio 8. Suponga que `C` es un tipo abstracto de datos cuya estructura de representación tiene dos campos de tipo `String`:

```
1 class C {
2 public:
3 /* ... */
4 private:
5     String s;
6     String t;
7 }
```

¿Cuál de los siguientes items podría ser parte del invariante de representación para `C`? Justificar.

- `s` sólo contiene letras.
- `s.length()` $=$ `t.length()`
- s representa un conjunto de caracteres.
- Los observadores de `C`.
- `s` es el reverso de `t`.
- `s` $+$ `t`

Ejercicio 9. Dado el siguiente tipo abstracto de datos junto con su estructura de representación:

```
class Palabras{
  public:
    // Constructor
    Palabras();

    // Observadores
    bool contiene(string p) const;

    // Modificadores
    void agregar_palabras(vector<string> ps);
    void borrar_palabra(string p);

    // Otras operaciones
    int contar_vocales() const;

private:
    vector<string> palabras;
}
```

$Rep(e : estr) \equiv (\forall i, j : \textbf{int})\,(0 \leq i < |e.palabras| \land 0 \leq j < |e.palabras[i]| \implies esLetra(palabras[i][j]))$

$esLetra(c : \textbf{char}) \equiv c$ es una letra del alfabeto.

(a) ¿Qué precondición debería tener el método `agregar_palabra` para garantizar que el invariante de representación se mantiene? En general, ¿cuál es el vínculo entre la precondición de los métodos y el invariante Rep?

(b) Implementar los métodos en C++ asumiendo las precondiciones que vea necesarias.

(c) ¿Cuál es el órden de complejidad O en peor caso para el método `contar_vocales`? Puede tomar como tamaño de entrada N la suma de todas las longitudes todas las palabras.

(d) ¿Se puede dar una implementación de `contar_vocales` con órden de complejidad $O(1)$ en peor caso, respetando la estructura de datos dada?

(e) Proponer una nueva estructura de datos con su invariante de representación que permita implementar `contar_vocales` en $O(1)$. Volver a implementar todos los metodos respetando la nueva estructura y el nuevo invariante de representación.