1. **(10 %)** A partir del análisis de los archivos “ DataSet.h”, “ DataSet.hxx”, “ DataSetCollection.h” y

“DataSetCollection.hxx”, realice en el informe la definición formal de los Tipos Abstractos de Datos (TADs) utilizados en el simulador.

Para cada TAD identificado, se debe presentar su especificación completa según el modelo explicado en clase, incluyendo:

* + Nombre del TAD
  + Descripción informal del propósito del TAD
  + Estado interno (atributos que lo definen)
  + Interfaz (lista de operaciones públicas con su tipo de entrada y salida, sin implementación)

Además, se debe incluir un diagrama de relación entre los TADs, en el que se evidencie la forma en que se componen o interactúan entre sí. La especificación debe reflejar fielmente lo que está implementado en el código. No se aceptarán definiciones genéricas o copiadas de internet.

TAD DataSet<T>: Representa un conjunto matemático finito de elementos únicos de tipo T y proporcionar operaciones clásicas de teoría de conjuntos.

**Estado:**

* Elements, (Elements), contenedor, almacena los valores únicos del conjunto.
* Nombre, (name), carácter, identifica el conjunto.

**Interface:**

* DataSet(setName:string) – Crea un conjunto.  
  Pre:ninguna.  
  Post: getName()=setName, size()=0.
* getName(): string – Obtiene el nombre.  
  Pre: ninguna.  
  Post: retorna name.
* setName(newName:string) – Cambia el nombre.  
  Pre: newname != “ ”.  
  Post: name=newName.
* insert(value:T) – Inserta value si no está.  
  Pre: ninguna.  
  Post: contains(value)=true.
* contains(value:T): bool – Devuelve true si se contiene un elemento dado.  
  Pre: ninguna.  
  Post: Si es true, value ∈ elements.
* unionWith(other:DataSet<T>): DataSet<T> - Devuelve todos los elementos sin duplicados.   
  Pre: other válido.  
  Post: {x | x∈this ∨ x∈other}.
* intersectionWith(other:DataSet<T>): DataSet<T> - Devuelve los elementos comunes.  
  Pre: other válido.  
  Post: {x | x∈this ∧ x∈other}.
* differenceWith(other:DataSet<T>): DataSet<T> - Devuelve los elementos que están en This y no en Other.  
  Pre: other válido.  
  Post: {x | x∈this ∧ x∉other}.
* symmetricDifferenceWith(other:DataSet<T>): DataSet<T> - Devuelve los elementos que están solo en uno.   
  Pre: other válido.  
  Post: {x | (x∈this) ⊕ (x∈other)}.
* isSubsetOf(other:DataSet<T>): bool – Devuelve true si todo element de This está en Other.  
  Pre: other válido.  
  Post: Si es true, x∈this, x∈other.
* isEqualTo(other:DataSet<T>): bool – Devuelve true si tienen la misma cardinalidad y mismos elementos.  
  Pre: other válido.  
  Post: Si es true, mismos elementos.
* size(): size\_t – Devuelve el tamaño.  
  Pre: ninguna.  
  Post: cardinalidad.
* getElements(): vector<T> - Copia y retorna elements.   
  Pre: ninguna.  
  Post: retorna copia de elements.
* print(os:ostream=cout): void –Imprime representación del conjunto.  
  Pre: os válido.  
  Post: escribe representación en os.
* powerSet(): DataSet<DataSet<T>> - Devuelve el conjunto de todos los subconjuntos.  
  Pre: ninguna.  
  Post: todos los subconjuntos de this.
* cartesianProductWith(other:DataSet<T>): DataSet<pair<T,T>> -   
  Pre: other válido.  
  Post: {(a,b) | a∈this ∧ b∈other}.
* operator==(a:DataSet<T>, b:DataSet<T>): bool – Devuelve rue si contienen los mismos elementos sin importar el orden.  
  Pre: This y Other válidos.  
  Post: Si es true, mismos elementos.

TAD DataSetCollection<T>: Representa una colección de conjuntos DataSet<T>, permite ejecutar operaciones clásicas entre ellos.

**Estado:**

* **Sets, (sets), contenedor,** almacena los conjuntos junto con su nombre único.

**Interface:**

* DataSetCollection() – Crea una colección vacía.  
  Pre: ninguna.  
  Post: sets != “ “.
* findIndexByName(name:string): int – Retorna la posición de un conjunto según su nombre. *(método privado)*  
  Pre: name != “ “.  
  Post: devuelve el índice del conjunto con ese nombre, o -1 si no existe
* addSet(set:DataSet<T>) – Agrega un conjunto.  
  Pre: set válido.  
  Post: si existe un conjunto con el mismo nombre, se reemplaza y si no, se agrega.
* hasSet(name:string): bool – Verifica si existe un conjunto con ese nombre.  
  Pre: name != “ “.  
  Post: Si es true, retorna un conjunto con nombre name.
* insertInto(name:string, value:T) – Inserta un valor en un conjunto identificado por nombre.  
  Pre: existe un conjunto con name.  
  Post: value pertenece al conjunto con nombre name.
* printSet(name:string): void – Imprime el conjunto identificado.  
  Pre: existe un conjunto con name.  
  Post: se muestra el conjunto con nombre name.
* getSet(name:string): DataSet<T> – Devuelve una copia de un conjunto identificado por nombre.  
  Pre: existe un conjunto con name.  
  Post: retorna una copia del conjunto name.
* getSetNames(): vector<string> – Devuelve los nombres de todos los conjuntos.  
  Pre: ninguna.  
  Post: retorna la lista de nombres almacenados en sets.
* operate(nameA:string, op:string, nameB:string): DataSet<T> – Ejecuta una operación binaria entre dos conjuntos.  
  Pre: nameA != “ “, nameB != “ “, y op = {union, intersection, difference, symmetricDifference}.  
  Post: devuelve el conjunto resultado de aplicar op entre nameA y nameB.
* operateUnarySet(name:string, op:string): DataSet<DataSet<T>> – Ejecuta una operación unaria sobre un conjunto.  
  Pre: name != “ “, op = powerset.  
  Post: devuelve el conjunto potencia de name.
* cartesianProduct(nameA:string, nameB:string): DataSet<pair<T,T>> – Producto cartesiano entre dos conjuntos.  
  Pre: nameA != “ “, nameB != “ “.  
  Post: (a,b) = {a ∈ conjunto(nameA) , b ∈ conjunto(nameB)}.

