Definiciones previas

Estado de la ejecución de un programa
Se define *estado de la ejecución de un programa* al conjunto de valores de todas sus variables en un instante dado. El estado inicial lo determinan los datos de entrada y el estado final los datos de salida.

Precondición y Postcondicion
Un programa, función, subrutina o método es una secuencia de instrucciones que hace que se transforme un estado inicial en un estado final. Las condiciones que deben cumplir los datos de entrada para que el programa ejecute correctamente se denomina *precondición* y las condiciones de los datos de salida se denomina *postcondición.*

Dominio y codominio
Dominio es el conjunto de datos sobre el que se aplica el método o función. Codominio es el conjunto de salida.

Especificación de una operación (función, método, subrutina)
La especificación de una operación consiste en:

- un nombre
- dominio y codominio
- comentario describiendo que es lo que hace
- precondicón
- postcondición

Notación
nombre: codominio → codominio
/* comentario si es necesario
precondición: ----
postcondición: ----

Ejemplo
Supongamos un método que calcula la raíz cuadrada de un número real. El dominio y el codominio son los números reales pero los valores de entrada deben verificar la precondición que deben ser mayores o iguales a cero en cuyo caso la postcondición es la raíz cuadrada del número.

raizCuadrada: R → R
/*Obtiene la raíz cuadrada de un número real
precondición: x € R / x >= 0
postcondición: $\sqrt{x}$

El comentario puede obviarse cuando el nombre es suficientemente claro de que es lo que hace la operación.
Para los nombres vamos a seguir las reglas de Java.

Ejercicios:
Especificar métodos que resuelvan los siguientes problemas:

1. Calcular el inverso de un número.
2. Calcular el factorial de un número.
3. Calcular el valor absoluto de un número real.

Cuando la operación tiene más de un dato de entrada indicamos el dominio como D1XD2X..XDn.
Idem con el codominio.

Ejercicios:
Especificar métodos que resuelvan los siguientes problemas:

1. Dadas las ecuaciones de dos rectas informar si son coincidentes, paralelas o se cortan en un punto.
2. Retornar el elemento que se encuentra en la i-ésima posición de un vector de N posiciones.
3. Hallar las raíces de la ecuación de segundo grado $a x^2 + bx + c = 0$.

Tipo Abstracto de Datos

¿Qué significa en Java o en C el tipo **int**?

En Java el tipo **int** es el conjunto de números enteros que pertenecen al rango [-2^{31}, $2^{31}-1$]; en C pertenecen al rango [-2^{15}, $2^{15}-1$]. No importa el lenguaje de programación a los elementos de este conjunto le podemos aplicar las operaciones: suma, resta, multiplicación, división y resto.

¿Qué significa en Java el tipo **String**?

El tipo **String** es el conjunto formado por cadenas de caracteres. En Java las operaciones o métodos que podemos aplicar a un elemento de tipo String son entre otros charAt(), concat(), equals(), length(), etc.

Para poder utilizar correctamente el tipo **int** solo es necesario conocer cuales son los posibles valores que puede almacenar una variable de tipo int y que operaciones podemos aplicar sobre ellos. No es necesario conocer, ni nos interesa saber cómo es el hardware que soporta a los números enteros ni cómo funciona el circuito sumador para sumar dos números enteros.

De la misma manera usamos el tipo **String** sin saber la forma en que las cadenas son almacenadas en memoria ni cual es el código de los métodos que aplicamos sobre ellas. Lo único que tenemos que saber es cómo usar estos métodos, que parámetros debemos pasar y que nos devuelve como resultado. En otras palabras cual es la funcionalidad y cual es la sintaxis. Por ejemplo para usar el método length() del tipo **String** alcanza con saber que a dicho método no debemos pasarle ningún parámetro y nos retorna un entero que representa la cantidad de caracteres de la cadena a que le aplicamos length().

El tipo **int** es provisto por cualquier lenguaje de programación. El tipo **String** es provisto por Java pero no por C. Otros lenguajes con fuerte orientación matemática proveen el tipo **Complex** para manipular números complejos. Ni Java ni C proveen este tipo de dato.

En cualquier caso no sabemos cómo están codificados y en tal caso se denominan ***Tipo Abstracto de Datos (TAD)***, ***Tipo de Dato Abstracto (TDA)*** o en inglés ***Abstract Data Type (ADT)***.

> Un tipo abstracto de datos es un conjunto de valores y una colección de operaciones definidas que nos permiten manipular esos valores.
>
> TAD = valores + operaciones

En la construcción de un TAD, debemos realizar tres pasos:

- Especificación
- Interface
- Implementación

La <u>especificación</u> es **independiente** del lenguaje de programación y de la forma en que los elementos del TAD se almacenan en memoria.

La <u>interface</u> es la firma o encabezamiento o sintaxis de cada operación en un lenguaje de programación. Consta del nombre del método, tipo de los parámetros de entrada y tipo del valor de salida. La interface **depende** del lenguaje elegido para programar el TAD pero no de la estructura elegida para representar los elementos del mismo.

La <u>implementación</u> consiste en la codificación de cada método en el lenguaje de programación. Para implementar un TAD debemos no solo haber elegido el lenguaje sino también la estructura de datos (variables simples, arreglos de una o dos dimesniones, nodos, etc) que usaremos para almacenar los elementos del TAD).

Un programador usuario de un TAD sólo necesita conocer la especificación y la interface.

La especificación consta de:

- Nombre del TAD
- Descripción en lenguaje natural, en lenguaje matemático o en un lenguaje gráfico según convenga de los elementos que pertenecen al TAD.
- Conjunto de operaciones denominadas operaciones primitivas. Cada operación se describe de la siguiente manera:
  - Nombre
  - Comentario si es necesario
  - Dominio y codominio
  - Precondiciones y poscondiciones

Las operaciones se clasifican en constructoras, modificadoras, analizadoras y destructoras.

- Las constructoras permiten crear nuevos elementos del TAD. Su objetivo es reservar lugar de memoria para guardar dichos elementos.
- Las modificadoras modifican el estado del elemento del TAD sobre la que se le aplica.
- Las analizadoras permiten obtener información sobre el estado del elemento sin modificar el estado del elemento al que se le aplica.
- Las destructoras destruyen elementos del TAD, entre otras palabras liberan memoria.

Ejemplo

Especificación del tipo abstracto de dato Diccionario

Nombre del TAD: Diccionario

Descripción: Un diccionario es un conjunto ordenado alfabéticamente de palabras, cada una de las cuales tiene uno o más significados.

Constructoras:

- **Diccionario: -** $\longrightarrow$ **Diccionario**
  - Construye un diccionario sin palabras, esta operación solicita memoria para poder almacenar el conjunto de palabras con sus significados.
  - precondición: -
  - postcondición: Diccionario d = ϕ

Modificadoras:

- **agregarPalabra : Diccionario X Palabra X Significado** $\longrightarrow$ **Diccionario**
  - Agrega al Diccionario ***d*** una nueva palabra ***pal*** con un significado ***s***.
  - precondición: ***pal*** no debe estar en el diccionario d.
  - postcondición: Diccionario ***d*** con la palabra ***pal*** y significado ***s***.
- **agregarSignificado: Diccionario x Palabra x Significado** $\longrightarrow$ **Diccionario**
  - Agrega un nuevo significado ***sig*** a la palabra ***pal*** en el Diccionario ***d***.
  - precondición: ***pal*** debe estar en el diccionario ***d***.
  - postcondición: Diccionario ***d*** con la palabra ***pal*** y significado ***s*** y ***n***.
- **eliminarSignificado: Diccionario x Palabra x Significado** $\longrightarrow$ **Diccionario**
  - Elimina el significado ***sig*** de la palabra ***pal*** en el Diccionario ***d***, si ***sig*** es el único significado de ***pal*** también debe eliminar ***pal***.
  - precondición: ***pal*** y ***sig*** debe estar en el diccionario d.
  - postcondición: Diccionario ***d*** con la palabra ***pal*** sin el significado ***sig*** o sin la palabra ***pal***.
- **eliminarPalabra: Diccionario x Palabra** $\longrightarrow$ **Diccionario**
  - Elimina la palabra ***pal*** con todos sus significados.
  - precondición: ***pal*** debe estar en el diccionario ***d***.
  - postcondición: Diccionario ***d*** sin la palabra ***pal***.

Analizadoras:

- **buscarPalabra: Diccionario X Palabra** $\longrightarrow$ **ListaDeSiginicados**
  - Obtiene todos los significados de la palabra ***pal***.
  - precondición**:** Palabra ***pal*** debe estar en el diccionario.
  - postcondición: sig_1, sig_2, ..., sig_n.

- **esVacio: Diccionario ⟶ boolean**
  - Informa si el diccionario ***d*** está o no vacío.
  - precondición: Diccionario ***d***.
  - postcondición: VERDADERO si $d = \phi$; FALSO si $d \neq \phi$.

Ejercicios

1. Especificar Número Complejo con las operaciones que permitan construir un número complejo y las analizadoras/modificadoras: asignaReal, asignaImagianrio, parteReal, parteImaginaria, módulo, suma, resta, producto y división.

2. Espeficiar el TAD Matriz de Números Reales con las operaciones que permitan construir una matriz de NxN y las analizadoras/modificadoras: asignar un valor en la posición i, j; obtener el valor de la posición i, j; sumar dos matrices, producto escalar; multiplicación de matrices.

3. Especificar una playa de estacionamiento.

Interface del TAD Diccionario

La interface en Java es:

```
public inteface Diccionario{
   void agregarPalabra (String  pal, String Significado);
   void agregarSignificado (String  pal, String Significado);
   void eliminarSignificado (String  pal, String Significado);
   void eliminarPalabra (String  pal, String Significado);
   ListaDeSiginicados buscarPalabra (String Palabra);
   boolean esVacio();
}
```

Ejercicio

Escribir las interfaces de los TAD anteriores.