



# Trabajo Práctico 1 – Smalltalk

[75.07/95.02/TB025] Algoritmos y Programación III  
Curso 01  
Primer cuatrimestre de 2025

Alumno:	KÖLLN, Dana Abigail
Número de padrón:	
Email:	

# Índice

1. Introducción	2
2. Supuestos	2
3. Modelo de dominio	3
4. Diagramas de clases	3
5. Detalles de implementación	6
Nacionalidad, País y Predicción	6
Registro, Cripto y Predicción	7
<b>6. Excepciones creadas</b>	<b>7</b>
<b>7. Diagramas de secuencia</b>	<b>8</b>

## 1. Introducción

El presente informe reúne la documentación de la solución del primer trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar una aplicación de un sistema para predicciones de criptomonedas, específicamente para calcular y registrar el valor de la crypto según características de la persona que recomienda. Esta fue desarrollada a partir de los conocimientos de la materia sobre el paradigma de la programación orientada a objetos adquiridos hasta la fecha.

Decidí enfocarme en hacer un modelo que cumpla con los requerimientos, antes que en uno que permita poner Strings y castearlo a SmallInteger en los datos de entrada.

El modelo tiene muchas cosas para mejorar, como el incumplimiento de las precondiciones

## 2. Supuestos

A continuación voy a enumerar los supuestos que utilicé:

- Un recomendador no puede predecir una crypto más de una vez con el mismo nombre, pero sí puede predecir otras criptos.
- Si dos predicciones valen lo mismo (sea máxima o mínima), y se pide el nombre de un predictor según el criterio, se devuelve el primero anotado.
- Una instancia de la clase Crypto puede tener el nombre que se quiera, no es necesario que comience con “\$”.
- El criterio no cambia a lo largo del programa.
- Si se ingresa como nacionalidad cualquier String que no sea ‘Argentina’, se lo cuenta como extranjero.

Para no complejizar la lectura del código, decidí no implementar verificaciones para los datos de entrada ni constructores por defecto, ya que me han dicho que no es lo que se busca con este trabajo práctico.

### 3. Modelo de dominio

Para crear la aplicación utilicé el estilo de desarrollo dado por la cátedra, conocido como 'Test Driven Development'. Esta cuenta con una API llamada AlgoPrediccionCripto a través de la cual vamos a estar registrando predicciones, solicitando sus valores y demandando el nombre del recomendador que cumpla con el criterio de búsqueda. Para comenzar, utilizamos un constructor para crear una instancia de AlgoPrediccionCripto con un criterio, este va a ser un atributo de la clase Criterio, la cual puede tener una categoría de la clase Mínimo o Máximo según se indique.

Al comienzo el diseño era sencillo de modelar gracias al flujo de desarrollo TDD, luego se fue complejizando a medida que iba refactorizando, pero el crear más pruebas ayudaba a visualizar mejor.

### 4. Diagramas de clases

En el diagrama de clases de la figura 1 podemos observar la relación de herencia que tienen Argentina y Extranjero con País, hablo más sobre esto en detalles de implementación.

La clase nacionalidad tiene un atributo de la clase abstracta País, al cual se le envía el mensaje modificar.

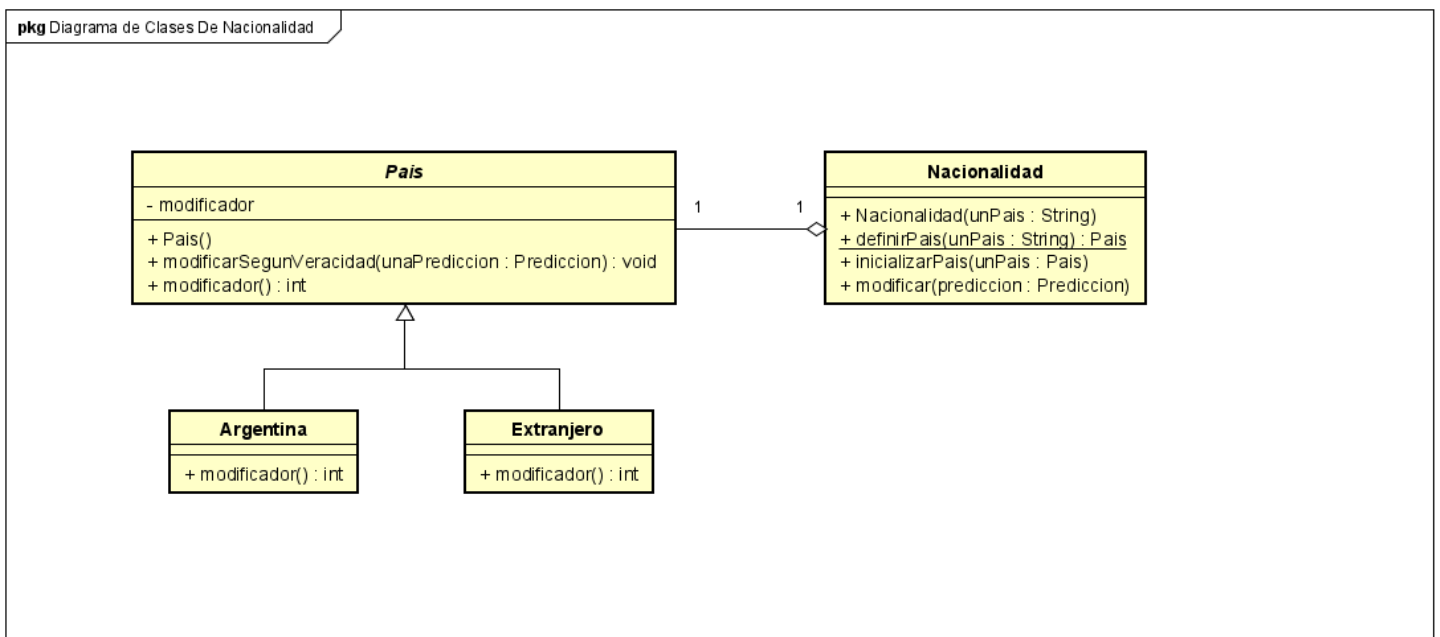


Figura 1: Diagrama de clase de Nacionalidad

En el diagrama de la figura 2 observamos la relación de la clase Cabello con las clases que heredan de la Interfaz Color.

Aclaración: no les reescribí el método a Colorado, Rubio y Morocho ya que se sobreentiende que cada uno tiene una implementación distinta de ese método.

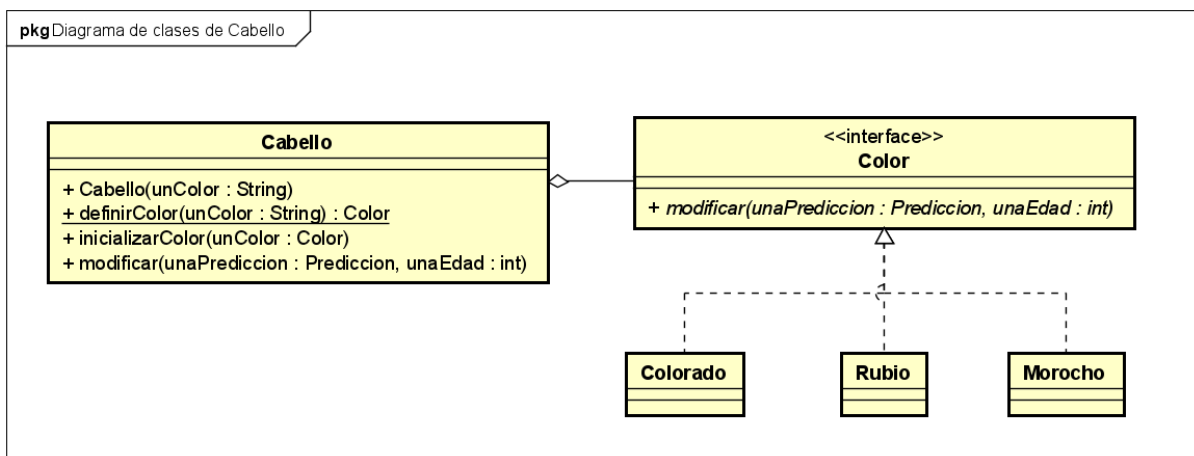


Figura 2: Diagrama de clase de Cabello

En la figura 3 tenemos la relación entre Registro, Cripto y Predicción. Una instancia de un registro va a tener una colección de objetos de la clase Cripto, que a su vez tiene una colección de objetos de la clase Predicción.

Podemos ver que Predicción tiene dos getters, pero son necesarios dados los tests de la cátedra.

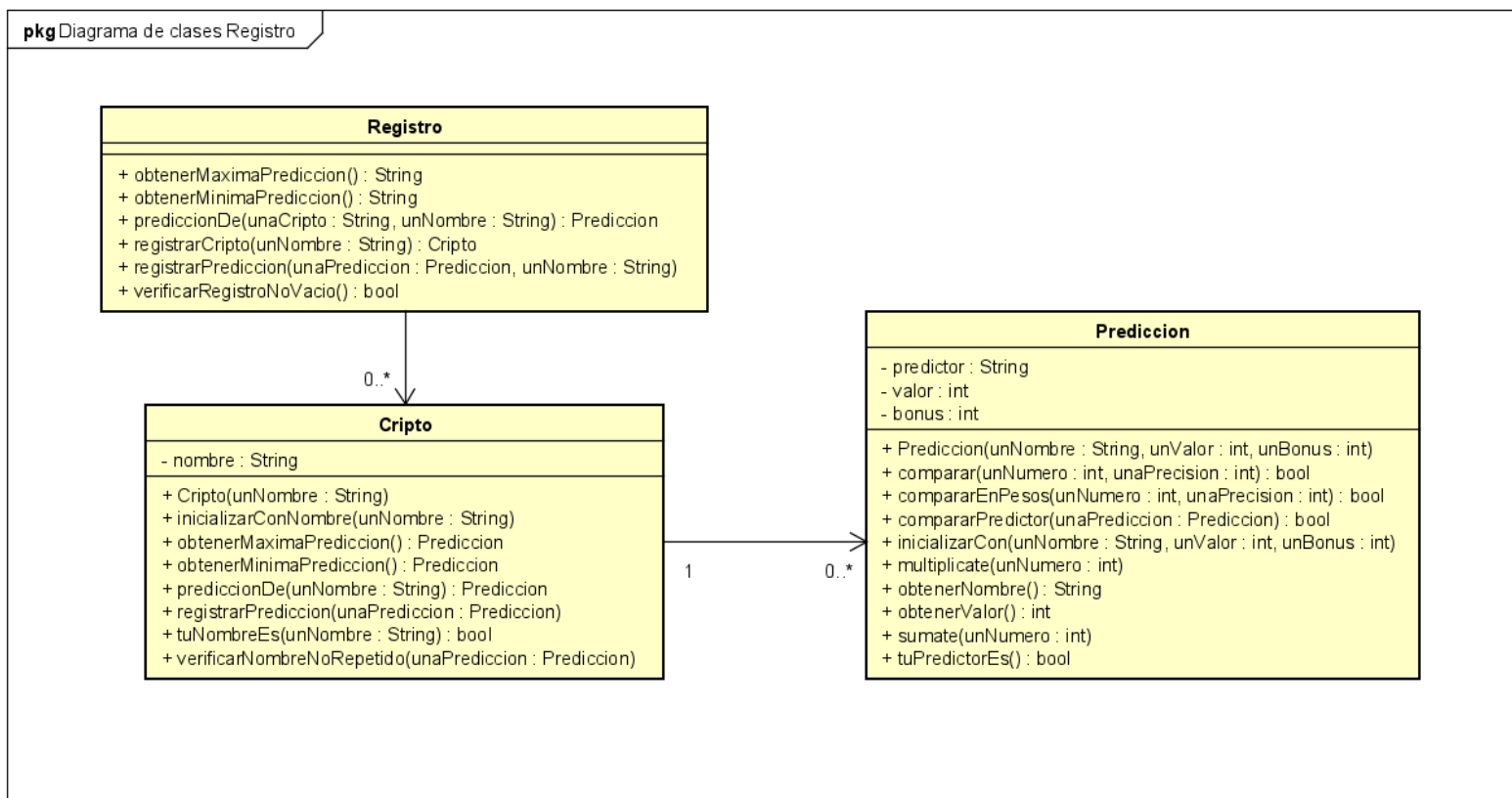


Figura 3: Diagrama de clases de Registro

En la figura 4 vemos la relación de AlgoPrediccionCripto con sus atributos.

Al crearse una instancia de esta clase se inicializa con un registro de la clase Registro y un creador de Predicciones. También se le especifica al constructor el criterio que se quiere, el cual se le delega a Criterio la definición de la categoría.

Como aclaré anteriormente, no vuelvo a escribir los métodos de la interfaz Categoría en sus hijos ya que se sobreentiende.

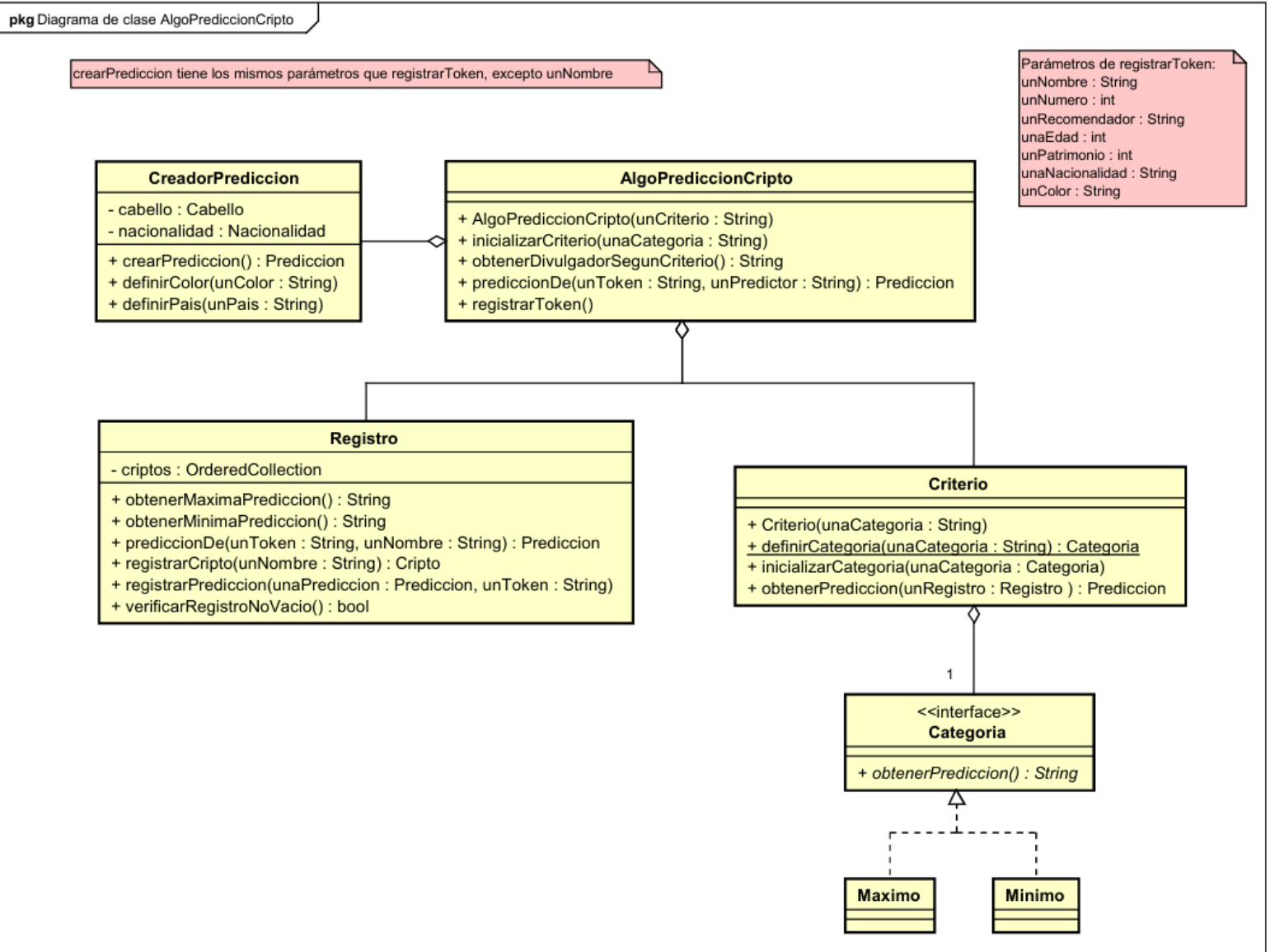


Figura 4: Diagrama de clase de AlgoPrediccionCripto

## 5. Detalles de implementación

### Nacionalidad, País y Predicción

Como podemos ver en el diagrama de clases de la figura 1, Nacionalidad le envía un mensaje a su atributo de la clase País, de la cual heredan Extranjero y Argentino. El mensaje enviado es `modificarSegunVeracidad` y cada clase le delega la modificación a la predicción de la clase Predicción, según su modificador, que es inicializado de diferente

forma en cada clase hija. De esta forma estamos ocultando cómo modifica cada país, y cómo se modifica internamente la predicción, cada uno sabe lo que debe hacer. Evitamos repetir código en Argentina y Extranjero.

Para inicializarse, Pais llama a un método llamado modificador, el cual devuelve una constante y se la pone como atributo al inicializarse la instancia. En Pais no está definida, es responsabilidad de las clases hijas.

## Registro, Cripto y Predicción

En el diagrama de la figura 3 podemos observar que en Predicción existen getters, pero son necesarios para los tests ya que nos piden obtener el nombre de la predicción según cierto criterio, o el valor de la predicción de cierta persona.

¿Por qué decidí separar en dos colecciones?

Para que sea más sencillo verificar mis supuestos.

Para acortar la búsqueda de una predicción.

Para darle una entidad a la cripto.

Delegar responsabilidades.

Para registrar una predicción de un token primero se busca si existe la cripto, en caso negativo se crea y luego se le delega el registro de la predicción.

Y como podemos ver en el siguiente código, para obtener la predicción según el criterio (en este caso mínimo), se guardan todas la predicciones mínimas de cada cripto en una colección y se busca la menor de ellas:

```
obtenerMinimaPrediccion
  |minimos minimaPrediccion|
  self verificarRegistroNoVacio.
  minimos := criptos collect:[:cripto | cripto obtenerMinimaPrediccion].
  minimaPrediccion := minimos detectMin:[:minimo | minimo obtenerValor].
  ^minimaPrediccion obtenerNombre.
```



## 6. Excepciones creadas

**ColorInvalidoError:** Si el color de cabello ingresado no se encuentra en las opciones, lanza esta excepción.

**CriptoNoEncontradaError:** cuando se le pide al registro que devuelva la predicción de cierta persona sobre una cripto que no está registrada, se lanza esta excepción .

**CriterioInvalidoError:** Al principio del programa se crea una instancia de la clase AlgoPrediccionCripto a la que se le pasa por parámetro un string que es el encargado de determinar el criterio. Si luego de transformarlo a minúsculas no coincide con ninguna de las opciones, se le hace saber a quien esté usando la aplicación que el string ingresado no es válido a través de esta excepción.

**PrediccionNoEncontradaError:** se lanza esta excepción cuando ningún objeto de la clase Cripto registrado tiene una predicción con el predictor pasado por parámetro.

**PredictorRepetidoError:** se lanza esta excepción para hacerle saber al usuario que el predictor con el nombre que ingresó ya fue registrado.

**RegistroVacioError:** se lanza cuando se pide algo del registro pero está vacío.

## 7. Diagramas de secuencia

Para los diagramas de secuencia voy a utilizar el test02 dado por la cátedra que es el siguiente:

```
test02UnRecomendadorExtranjeroConCiertColorDePeloPatrimonioYEdadPuedePredecirUnAumento
DeCiertaCriptoDeCiertValor
    | prediccionEsperada prediccionObtenida algoPrediccionCripto |

    "Arrange"
    prediccionEsperada := 314986000.
    algoPrediccionCripto := AlgoPrediccionCripto conCriterio: 'Maximo'.

    algoPrediccionCripto registrarTokenConNombre: '$MAGA' ConValorActual: 1000
    ConRecomendador: 'Donald' yEdad: 78 yPatrimonio: 300000000 deNacionalidad: 'Estados
    Unidos' yColorDePelo: 'Rubio'.

    "Act"
    prediccionObtenida := algoPrediccionCripto prediccionDe: '$MAGA' conRecomendador:
    'Donald'.

    "Assert"
    self assert: (prediccionObtenida compararCon: prediccionEsperada precision:
    precision )
```

En la figura 5 tenemos la secuencia de cómo se crea la predicción muy por arriba.

Aclaraciones:

- apc : algoPrediccionCripto
- cp : creadorDePredicciones
- n : nacionalidad
- rc : registroCriptos

Los parámetros que se le pasan a mensaje registrarToken son:

('MAGA',1000,'Donald',78,300000000,'Estados Unidos','Rubio')

y a crearPrediccion:

(1000,'Donald',78,300000000,'Estados Unidos','Rubio')

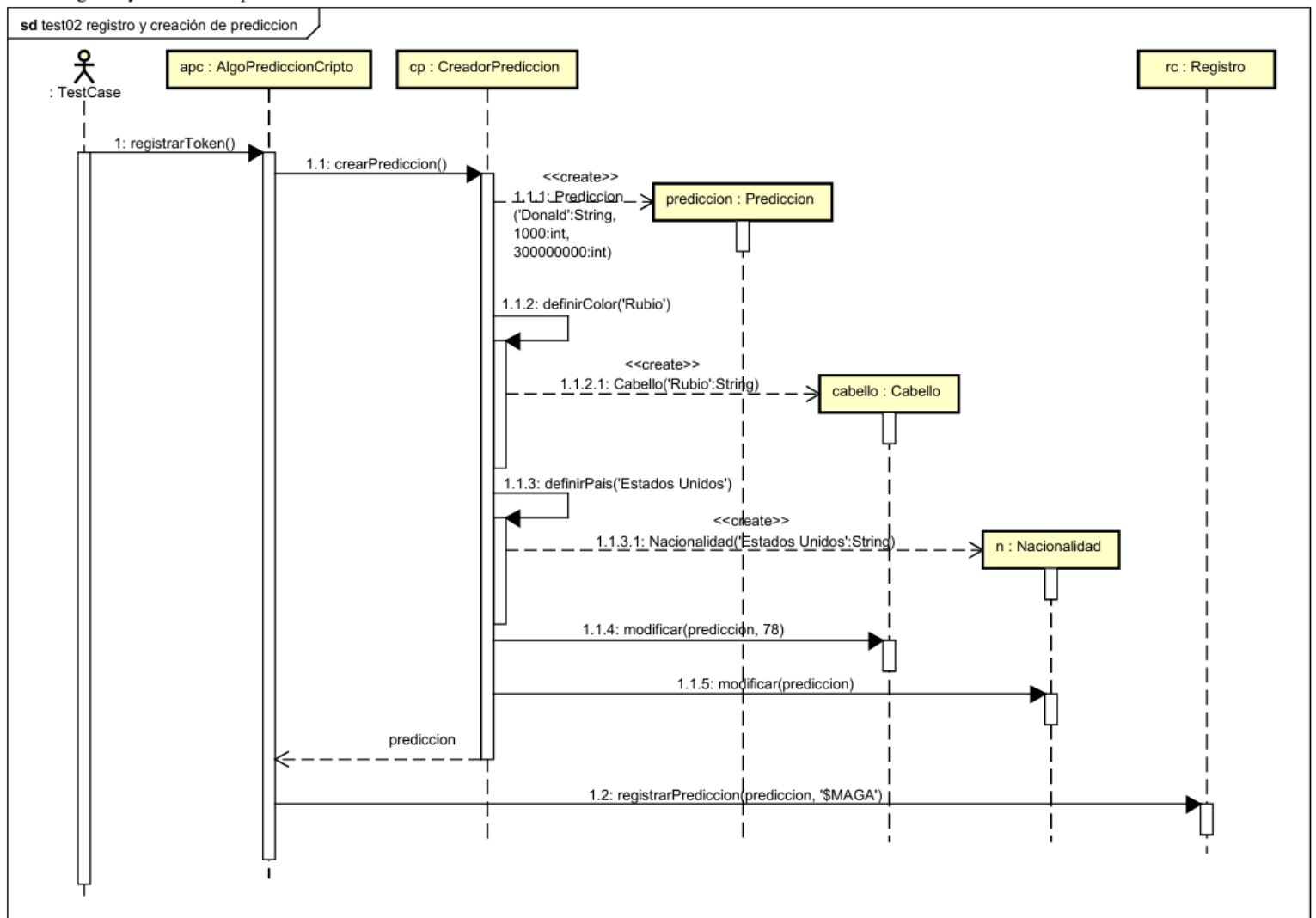


Figura 5: Diagrama de secuencia test02 creación y registro de predicción

En la figura 6 podemos ver cómo se va delegando la tarea de encontrar la predicción. Vemos que se le envía a una instancia de la clase `OrderedCollection` el mensaje `detect`, el cual hace un recorrido por cada cripto y devuelve la que cumple con el bloque especificado en el cuadro rosa.

Como quedaba demasiado largo decidí mostrar por arriba la invocación del mensaje que se le envía a la cripto buscada. Dentro de la misma hay otra iteración por la colección “predicciones”, que tiene como atributo, y devuelve la que cumple con la condición.

Luego, nuestro actor le envía un mensaje a la misma con un valor, para verificar que es la correcta.

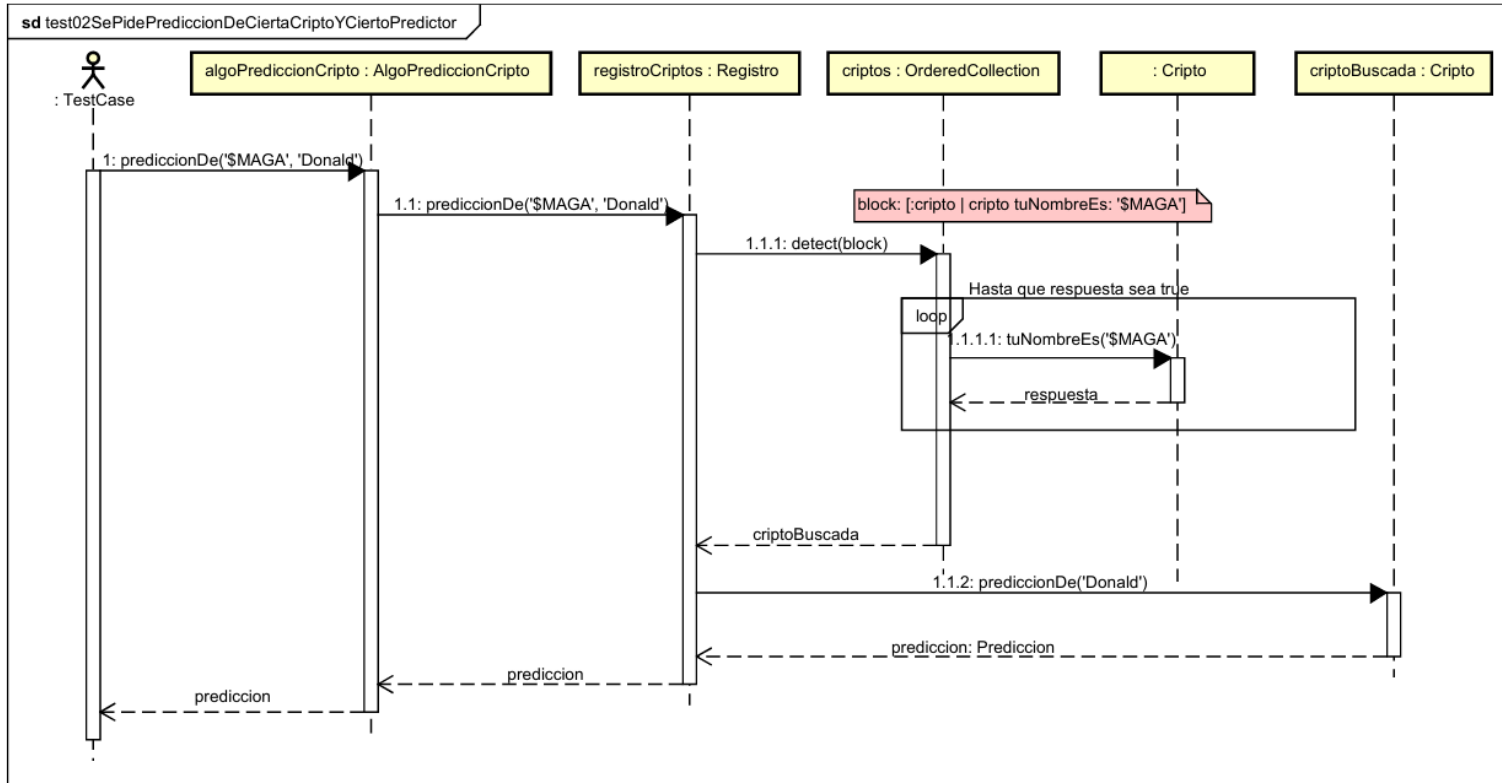


Figura 6: Diagrama de secuencia segunda parte del test02