**Oracle**

**Identificación del problema**

En la etapa de desarrollo que está realizando la compañía Oracle se encuentra en una de las ideas lanzar una nueva funcionalidad dentro de Java. Decidieron implementar una funcionalidad para encontrar las raíces de un polinomio. Esta decisión fue tomada gracias a la recolección de datos y encuestas de los IDE’s que muestra que los desarrolladores necesitan este tipo de funcionalidades ya que pueden ser de gran ayuda para estudiantes de ingeniería.

Por eso Oracle ha decidido implementar el desarrollo de esta funcionalidad donde se pueda entender, seleccionar e implementar como mínimo 2 algoritmos para encontrar raíces de polinomios. Oracle espera que se argumente claramente cuál fue el proceso de selección que se tuvo para elegir esos algoritmos de búsqueda de raíces.

**Recopilación de la información**

**Polinomio:**

En matemáticas, un polinomio [...] es una expresión algebraica constituida por una suma finita de productos entre variables (valores no determinados o desconocidos) y constantes (números fijos llamados coeficientes), o bien una sola variable. Las variables pueden tener exponentes de valores definidos naturales incluido el cero y cuyo valor máximo se conocerá como grado del polinomio. En términos más simples, un polinomio se toma como una suma de monomios, pero un monomio también se toma como un polinomio. (Wikipedia, 2019)

**Grado:**

En álgebra grado de un polinomio es el grado máximo de los exponentes de las variables de los monomios que lo componen. *Grado* tiene básicamente el mismo significado cuando se refiere a un polinomio o a una ecuación algebraica.En consecuencia, la primera definición que quizás deba revisarse sea la del monomio, considerado por el Álgebra elemental como una expresión algebraica básica, la cual se encuentra conformada por una combinación de números y letras (elevadas a exponentes enteros y positivos, incluido el cero) entre los cuales no caben operaciones de resta, suma o división, siendo entonces las únicas permitidas, la multiplicación planteada entre el elemento numérico (coeficiente) y el elemento no numérico (literal o variable) así como la potenciación ocurrida entre el literal y su exponente.

**Coeficiente:**

En matemáticas, un coeficiente es un factor vinculado a un monomio. Dado un divisor del monomio, el coeficiente es el cociente del monomio por el divisor. Así el monomio es el producto del coeficiente y el divisor. Los diferentes coeficientes dependerán de la factorización del monomio. Este suele estar junto a la letra que acompaña a la fracción algebraica. Un coeficiente numérico es un factor constante de un objeto específico.

**Raíz de polinomio:**

Las raíces de un polinomio son números tales que hacen que un polinomio valga cero. Podemos decir también que las raíces enteras de un polinomio de coeficientes enteros serán divisores del término independiente.

**Término independiente:**

Valor del polinomio el cual no va acompañado por un expresión(variable). en un polinomio ordenado es el último valor del polinomio; Este término puede o no estar en un polinomio.

**Monomio:**

Expresión algebraica que consta de un solo término o en que los términos que la forman están relacionados por la operación producto.

**Búsqueda de soluciones creativas**

Para resolver el problema de las raíces de un polinomio vamos a utilizar diferentes algoritmos y posteriormente se seleccionarán los mejores mediante unos criterios definidos posteriormente.

**Especificación de requerimientos funcionales**

**R1**. Encontrar las raíces de un polinomio:

El programa permite encontrar los raíces o ceros de un polinomio de hasta grado 10 sean las soluciones reales o imaginarias.

**R2.** Generar un polinomio con valores aleatorios:

Se genera un polinomio con valores aleatorios dado el valor del grado del polinomio deseados por el usuario.

**R3.** Editar el polinomio generado aleatoriamente:

Dado el polinomio con valores aleatorios el usuario podrá modificar los valores aleatorios por los que él quiera.

R4. Guardar el polinomio generado:

El programa permite guardar los polinomios generados por el usuario.

**R5.** Solucióna del polinomio:

El programa encontrará las raíces de polinomio con dos algoritmos diferentes usando el que mejor lo resuelva. Esto depende de la complejidad del polinomio.

**R6.** Restricción en la escritura de un polinomio:

Dependiendo del grado del polinomio deseado por el usuario el programa habilita las casillas necesarias para evitar que se generen errores.

**R7.** Solucionar algoritmo de grado 10:

Permitir ingresar un polinomio de grado 10, y generar la solución del polinomio de manera correcta y eficiente.

**R8.** Mostrar raíces:

Se debe permitir al usuario ver las raíces que dan solución a los polinomios.

**Especifica de requerimientos no funcionales**

**RNF1.** La solución debe ser correcta:

La solución debe acordar con los resultados correctos para el polinomio dado a solucionar.

**RNF2.** Solucionarse de manera eficiente:

Las raíces del algoritmo deben solucionarse una manera eficiente que no requiera mucho tiempo y recursos.

**RNF3.** El programa debe ser entendible y fácil acceso:

La interfaz que controla el programa no sede ser complicada para el uso.

**Búsqueda de soluciones creativas**

**Alternativa 1. Método de Graeffe:**

Se utiliza para calcular todas las raíces de una ecuación sean reales o imaginarias. Es el ùnico mètodo práctico para calcular raíces imaginarias. Este método resuelve varios casos que se puedan presentar. (.Ehu, 2016)

Sea el polinomio

a0xn+a1xn−1+a2xn−2+a3xn−3+ ... an−1x+an=0  (1)a0xn+a1xn−1+a2xn−2+a3xn−3+ ... an−1x+an=0  (1)

Hacemos el polinomio más simple dividiendo todos los coeficientes por el primer término de modo que a0 es siempre 1. Supongamos que sus raíces reales y distintas son

-r1, -r2, -r3, ...-rn.

Al elevar al cuadrado el polinomio y agrupar los términos se obtiene un polinomio de grado 2n

a20(xn)2−(a21−2a2a0)(xn−1)2+(a22−2a1a3+2a4a0)(xn−2)2−(a23−2a2a4+2a1a5−2a6a0)(xn−3)2+ ...=0}

(2)a02(xn)2−(a12−2a2a0)(xn−1)2+(a22−2a1a3+2a4a0)(xn−2)2−(a32−2a2a4+2a1a5−2a6a0)(xn−3)2+ ...=0} (2)

Cuyas raíces serán

−r21,−r22,−r23 ... −r2n

**Alternativa 2. Algoritmo de Horner:**

Es un [algoritmo](https://es.wikipedia.org/wiki/Algoritmo) para evaluar de forma eficiente [funciones polinómicas](https://es.wikipedia.org/wiki/Polinomio) de una forma [monomial](https://es.wikipedia.org/wiki/Monomio). Aunque la solución de un polinomio para un valor específico de x es una tarea sencilla el algoritmo reduce la cantidad de operaciones necesarias para llegar al resultado lo que la convierte en una técnica más eficiente y más deseable a la hora de programarla. Llamando a el grado del polinomio g una resolución por sustituciones requiere hasta (g2+g)/2 multiplicaciones y g sumas mientras que el algoritmo de Horner solo requerirá g sumas y g multiplicaciones.

**Alternativa 3.Algoritmo de Sturm:**

Supongamos que se van a buscar los ceros de un polinomio f (x), lo primero que se tiene que buscar al utilizar el Teorema de Sturm es un sistema de polinomios especial para poder evaluar en él, veamos cómo calcular este sistema. El primer polinomio es el propio f (x); elsegundo polinomio es f'(x), es decir, la derivada del polinomio f (x), a este polinomio lo vamos a denotar f1 (x); para el tercer polinomio se debe realizar la división de f (x) entre f1 (x), elresiduo que se obtiene con signo contrario será f2 (x); este proceso se continúa, es decir, f3 (x) será elresiduo con signo contrario de la división de f1 (x) entre f2 (x); el proceso acaba cuando se obtiene una constante.

**Alternativa 4. Algoritmo de Müller:**

Este método utilizado para encontrar raíces de ecuaciones con raíces múltiples, y consiste en obtener los coeficientes de la parábola que pasa por tres puntos elegidos. Dichos coeficientes son sustituidos en la formula cuadrática para obtener el valor donde la parábola intersecta al eje X; es decir, la raíz estimada. La aproximación se puede facilitar, si se escribe la ecuación de la parábola en una forma conveniente.

Una de las mayores ventajas de este método, es que al trabajar con la formula cuadrática es posible localizar tanto raíces reales, como raíces complejas.

**Alternativa 5. Método de Bairstow:**

En análisis numérico, el método de Bairstow es un algoritmo eficiente debúsqueda de las raíces de un polinomio real de grado arbitrario. Elalgoritmo apareció por primera vez en el apéndice del libro"Aerodinámica Aplicada", escrito por [Leonard Bairstow](http://es.wikipedia.org/wiki/Leonard_Bairstow) y publicado en1920. El algoritmo se diferencia de otros métodos en que encuentratanto las raíces reales como las imaginarias (en parejas complejasconjugadas), utilizando únicamente aritmética real.

**Alternativa 6. Método de Newton-Raphson:**

El método de Newton-Raphson es un procedimiento algorítmico que permite hallar raíces de funciones, conocido uN valor numérico cercano a la raíz. Es un método iterativo, en general de rápida convergencia, muy útil para el cálculo de raíces cuadradas y de mayor grado.

**Transición de las ideas a los diseños preliminares**

Beneficios de cada algoritmo en diferentes aspectos que serán útiles para la selección de los dos algoritmos que se utilizarán en el programa final.

Alternativa 1. Algoritmo de Graeffe:

* permite encontrar raíces de un polinomio sean reales o imaginarias.
* Es el único método práctico para calcular raíces imaginarias.
* Proporciona gran exactitud en las raíces de un polinomio.

Alternativa 2. Algoritmo de Horner:

* Método rápido y efectivo.
* Evalúa de forma eficiente las funciones polinómicas de una forma monomial.
* Minimizar el número de multiplicaciones es lo más deseable porque necesitan mucha carga computacional y son inestables comparadas con la suma.
* Se pueden resolver las divisiones con cualquier grado del polinomio.

Alternativa 3. Algoritmo de Sturm:

* Es útil para hallar los ceros de una función polinómica en un determinado intervalo.
* son muy pocos polinomia de raíces múltiples que fallan por el teorema de Sturm.

Alternativa 4. Algoritmo de Müller:

* Se encuentran tanto raíces reales como complejas.
* La principal desventaja es que en cada iteración se descarta una posible raíz de la parábola sin conocer la naturaleza de la misma.
* Converge cuadráticamente en un intervalo cercano a la raíz.
* Requiere valores iniciales.

Alternativa 5. Método de Bairstow:

* es un algoritmo eficiente de [búsqueda](https://es.wikipedia.org/wiki/Resoluci%C3%B3n_num%C3%A9rica_de_ecuaciones_no_lineales) de las [raíces](https://es.wikipedia.org/wiki/Ra%C3%ADz_de_una_funci%C3%B3n) de un [polinomio](https://es.wikipedia.org/wiki/Polinomio) real de grado arbitrario.
* Es un método iterativo.
* Para empezar, debemos de partir de dos polinomios cuadráticos.

Alternativa 6. Algoritmo de Newton-Raphson:

* Es un [algoritmo](https://es.wikipedia.org/wiki/Algoritmo) para encontrar aproximaciones de los [ceros o raíces](https://es.wikipedia.org/wiki/Ra%C3%ADz_de_una_funci%C3%B3n) de una función real.
* Es un método iterativo.
* Garantiza una aproximación exacta de la raíz.
* La iteración debe empezarse con un valor cercano a cero denominado punto de arranque.

**Evaluación y selección de la mejor solución**

Criterios para tomar la mejor solución para resolver un polinomio de grado menor o igual a 10.

Criterio A: Raíces reales y complejas.

* el algoritmo proporciona soluciones en el conjunto de los números reales como de los complejos

Criterio B: Exactitud la solución.

* Proporciona una respuesta más próxima a la respuesta exacta.

Criterio C: Efectividad del algoritmo para polinomio mayor a grado 2.

* Solución correcta y optima polinomios mayores a grado 2.

Criterio D: Tipo de algoritmo.

* Algoritmos iterativos o recursivos.

**Problema:** Raíces de un polinomio.

**Entradas:** Número n tal que n <= 10 siendo el grado del polinomio.

Coeficientes del polinomio P(x).

**Salida:** Número(s) n tal que P(n) = 0.

**Consideraciones**

Se deben tener en cuenta los siguientes casos al calcular las raíces de un polinomio:

1. **0 es raíz** cuando el polinomio con el que trabajamos no tiene término independiente. Es decir, cuando todos los términos de nuestro polinomio contienen alguna “x” cualquiera sea su exponente.
2. **1 es raíz del polinomio** cuando al sumar sus coeficientes obtenemos como resultado 0. Esto quiere decir que, si sumas los coeficientes que multiplican a tus “x” y también al término independiente y obtienes 0 como resultado, puedes asegurar que 1 es raíz.
3. **– 1 es raíz del polinomio** cuando la diferencia entre los coeficientes pares y los impares es 0.
4. **Si es de Grado 1:** Bastará con despejar la incógnita, x.
5. **Si es de Grado 2:** Resolveremos la ecuación de segundo grado resultante.
6. **Si es Bicuadrada:** Se trata de un polinomio de grado 4 incompleto, con sólo los términos de los grados 4, 2 y 0. Esta ecuación se puede transformar en una de segundo grado
7. **Si es de Grado mayor que 2, no bicuadrada:** En este caso, la regla de Ruffini será útil para encontrar las raíces enteras del polinomio.
8. **Si tenemos un producto de varios polinomios:** No es necesario realizar el producto de ellos para ver el polinomio final del que se trata y calcular después sus raíces. Sólo hay que tener en cuenta lo siguiente: Para que un producto sea 0, alguno de los factores debe ser cero.

**Evaluación para l selección de la mejor solución**

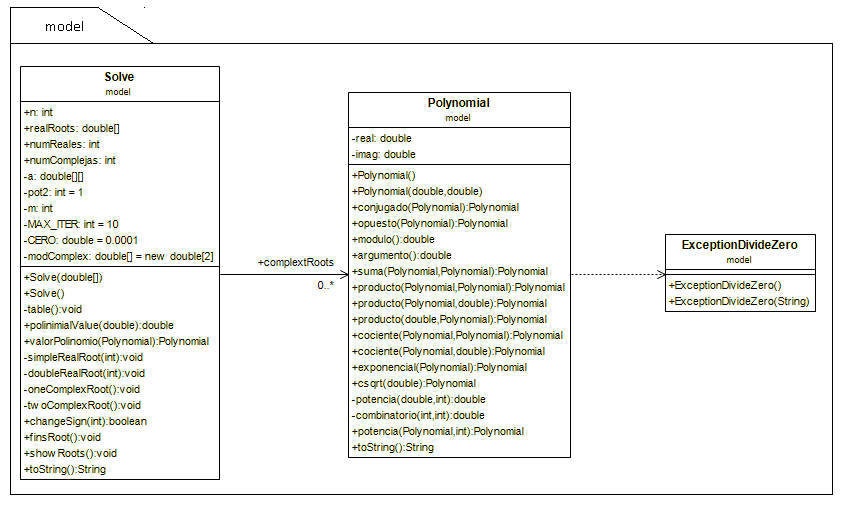
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Criterio A | Criterio B | Criterio C | Criterio D | Efectividad |
| Alternativa 1 | Permite raíces reales o complejas | Proporciona exactitud en la solución del problema | Es efectivo para polinomios de hasta grado 7. | Usa métodos iterativos | 7 |
| Alternativa 2 | Permite raíces reales. | Proporciona exactitud en raíces reales. | Es efectivo para polinomios de hasta grado 7. | Usa métodos iterativos minimizando el número de multiplicaciones | 7 |
| Alternativa 3 | Permite raíces reales. | Proporciona exactitud en raíces reales. | Es efectivo para polinomios en diferentes intervalos | Usa métodos iterativos | 5 |
| Alternativa 4 | Permite raíces reales o complejas | Proporciona exactitud en la solución del problema. | Es efectivo para polinomios de hasta grado 7. | Usa métodos iterativos | 6 |
| Alternativa 5 | Permite raíces reales. | Proporciona exactitud en raíces reales. | Es efectivo para polinomios de hasta grado 7. | Usa métodos iterativos | 5 |
| Alternativa 6 | Permite raíces reales. | Proporciona exactitud en la solución del problema. | Es efectivo para polinomios de hasta grado 7. | Usa métodos iterativos | 6 |

Nota: La efectividad para la solución del problema esta evaluada con un criterio de 1 a 10 donde 1 es poco efectivo y 10 la mayor efectividad en la solución.

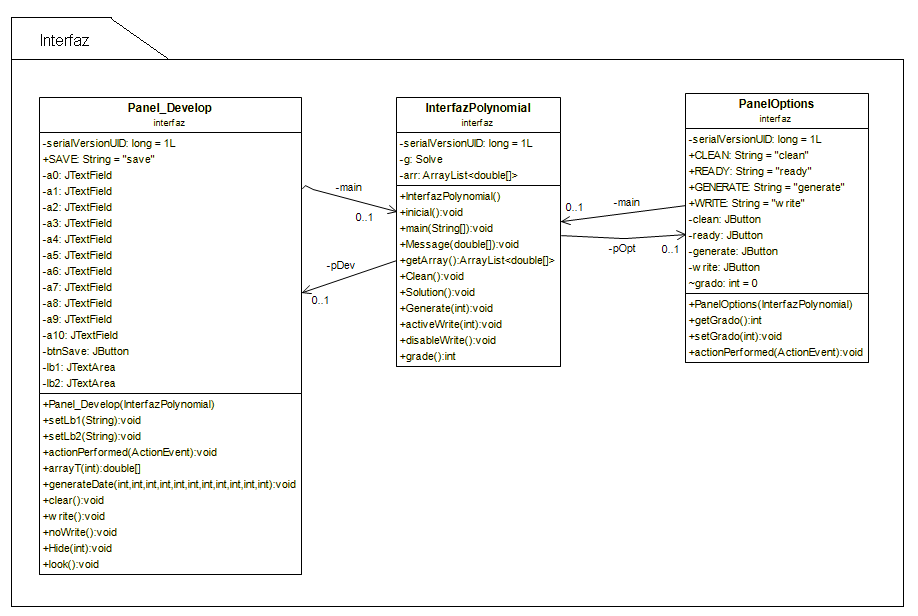
Para la solución del problema presentado el uso de las alternativas 1 y 2 dado que se la implementación y las propiedades de las alternativas están mas acordes a lo que se necesita para la solución correcta y efectiva del problema.

**Diseño Implementación**

**Diagrama de clases**



**Diagrama de la interfaz**



**Diseño Pruebas Unitarias**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| DISEÑO DE CASOS PARA PRUEBAS UNITARIAS | | | | | |
| Entrada | **Salida** | **Descripción** | **Nombre del método** | **Clase** | **Escenario** |
| {1,2,1} | 2 | Solución al problema con la cantidad requerida de raíces. | solve() | Solve() | sceneSolve() |
|  |  | Cantidad de raíces que tiene el polinomio | findRoots() | Solve() | sceneOne () |
|  |  | Cantidad de raíces que tiene el polinomio | polinomialValue() | Solve() | sceneTwo() |
|  |  | Cantidad de raíces que tiene el polinomio | table() | Solve() | sceneThree() |