El versionado semántico es un sistema utilizado para asignar números de versión a los software de manera que refleje los cambios que se hacen en la API pública de forma clara y predecible. Este sistema utiliza tres números separados por puntos (X.Y.Z) para indicar los cambios principales (X), cambios menores (Y) y correcciones de errores (Z), lo que facilita a los desarrolladores y usuarios entender cómo estos cambios afectan al software. Asimismo, algunos desarrolladores emplean cuatro dígitos, siendo el cuarto lo que se llama número de compilación.

**Especificaciones del versionado semántico:**

1. **Declarar una API pública:** Todo software que usa versionado semántico debe declarar su API pública, que puede estar dentro del código o en la documentación. Esta declaración debe ser precisa y completa para que los desarrolladores sepan cómo interactuar con el software.
2. **Formato de la versión:** El número de versión debe seguir el formato X.Y.Z, donde X es la versión mayor, Y es la versión menor y Z es la versión de parches. Estos números deben ser enteros sin ceros iniciales y siempre deben aumentar de forma secuencial (por ejemplo, 1.9.0 -> 1.10.0 -> 1.11.0).
3. **No modificar versiones publicadas:** Una vez que una versión ha sido lanzada, no debe ser modificada. Si se necesitan realizar cambios, estos deben ser lanzados como una nueva versión.
4. **Versión mayor cero (0.y.z):** Mientras un software esté en la versión 0.y.z, está en desarrollo inicial. Todo puede cambiar sin previo aviso, y no se debe considerar la API pública como estable.
5. **Versión 1.0.0:** La versión 1.0.0 define la primera API pública estable. A partir de este punto, cualquier cambio en la versión se reflejará en cómo cambia la API pública.
6. **Incremento de versión de parches:** El número Z (parches) debe incrementarse solo si se corrigen errores que no rompen la compatibilidad hacia atrás. Un parche es cualquier corrección interna que soluciona comportamientos incorrectos.
7. **Incremento de versión menor:** El número Y (versión menor) debe incrementarse cuando se agregue funcionalidad nueva que no rompa la compatibilidad hacia atrás. También debe incrementarse si se depreca alguna funcionalidad en la API pública. Además, puede incluir cambios de nivel de parche, y el número Z se reinicia a 0 cuando Y se incrementa.
8. **Incremento de versión mayor:** El número X (versión mayor) debe incrementarse cuando se introduzcan cambios en la API pública que no sean compatibles con versiones anteriores. También puede incluir cambios menores y de parches, y tanto Y como Z se reinician a 0 cuando se incrementa X.
9. **Versiones preliminares (pre-release):** Una versión preliminar puede indicarse añadiendo un guion y una serie de identificadores (como “alpha” o “beta”) después de la versión de parche. Estas versiones son inestables y pueden no ser totalmente compatibles con versiones anteriores. Por ejemplo, 1.0.0-alpha es una versión preliminar.
10. **Metadatos de compilación:** Los metadatos de compilación pueden agregarse añadiendo un signo de más (+) seguido de una serie de identificadores. Estos identificadores se usan para información adicional y no afectan el orden de precedencia de las versiones. Por ejemplo, 1.0.0+build20130313 no tiene prioridad sobre 1.0.0.
11. **Precedencia de versiones:** La precedencia indica cómo se comparan las versiones. Se determina al comparar los números de versión mayor, menor y de parches en ese orden. Si estos son iguales, las versiones preliminares tienen menor precedencia que las versiones normales. Dentro de las versiones preliminares, la precedencia se determina comparando cada identificador (numéricos se comparan como números y los alfanuméricos de manera léxica).

**19-09-2024**

**Explicación de los cambios**

1-. Se establece la versión inicial de la calculadora dentro de la rama **estable** como la versión **1.0.0.** Esta versión contiene una calculadora que permite realizar las siguientes operaciones básicas entre dos **números enteros:** suma, resta, multiplicación y división.

2-. Se cambia a la rama **master** para empezar a desarrollar las funcionalidades correspondientes para que la calculadora pueda operar con **números reales.** Se crea la versión **1.1.0** en la rama **master.** Dicha versión es una copia **exacta** de la versión **1.0.0** de la rama **estable**

3-. Se añade la versión **1.1.1** a la rama **master.** Esta versión añade las funcionalidades de **suma** y **resta** entre **dos números reales.**

4-. Se añade la versión **1.1.2** a la rama **master.** Esta versión añade las funcionalidades de **multiplicación** y **división** entre **dos números reales.**

5-. Durante el desarrollo de la versión **1.1.2** se detecta un error que provenía de la versión **1.0.0**. Este error se trata de que el programa no manejaba de forma correcta la división entre **cero.** Como dicho error se encontraba en la rama **estable,** se añade la versión **1.0.1** para corregirlo.

6-. Se corrige el error de **división por cero** en la rama máster. Para ello, se crea la versión **1.1.3**, cuyo código es idéntico al de la versión **1.1.2**, con la diferencia de que tiene el error de **división por cero** corregido.

7-. Como ya se ha finalizado la implementación de las **cuatro** operaciones básicas para **números reales**, publicamos la versión **2.0.0** en la rama **estable**. El código de dicha versión será **idéntico** al de la versión **1.1.3** de la rama **master**. Hemos incrementado el número mayor puesto que, la implementación de las operaciones con números reales ha supuesto un cambio en la **API** del código que existía en la anterior versión estable. Esto es así puesto que, al tratarse JAVA de un lenguaje **fuertemente tipado,** al cambiar la signatura y el tipo devuelto de los métodos Suma, Resta, Multiplicación y División, cualquier usuario que haya escrito un código que llamase a la versión anterior de la Calculadora, dicho usuario va a tener que modificar su código ya existente teniendo en cuenta los cambios realizados en la nueva versión para evitar así **los** **errores de compilación.**

8-. A continuación se procede a añadir las funcionalidades pertinentes para poder realizar las operaciones de **exponenciación** (las raíces cuadradas, raíces cúbica,… se entenderán como potencias con **exponente fraccionario**). Se crea la versión **2.1.0** en la rama **master**, cuyo código es **una copia** de la versión **2.0.0**.

9-. Se añade la versión **2.1.1** en la rama **master.** Dicha versión añade la funcionalidad de realizar la **exponenciación** entre dos números reales.

10-. Como ya hemos terminado la funcionalidad de la **exponenciación**, la publicamos en la rama **estable** en la versión **2.2.0**. El código de esta versión es **idéntico** al de la versión **2.1.1**.

11-. La siguiente funcionalidad a añadir sería la de introducir un parámetro opcional para controlar la **precisión** del resultado obtenido en las operaciones de **multiplicación**, **división** y **exponenciación.** Añadimos la versión **2.3.0** en la rama **master**. Aumentamos el **número menor** y no el **número mayor** puesto que la funcionalidad de la **exponenciación** no ha supuesto ningún cambio ni en el **API** ni en el **ABI.**

12-. Se crea la versión **2.3.1** en la que se incluye la precisión para la **multiplicación** de dos números reales.

13-. Se crea la versión **2.3.2** en la que se incluye la precisión para la **división** de dos números reales.

14-. Se crea la versión **2.3.3** en la que se incluye la precisión para la **exponenciación** de dos números reales.

15-. Un desarrollador del equipo de testing detecta un error en la lógica de la exponenciación: no se pueden hacer operaciones donde la base es **negativa** y el exponente es un número no entero. Se corrige el error en la **rama** estable en la versión **2.2.1**.

16-. Se corrige el error en la rama **estable** en la versión **2.3.4**