Semantic Versioning - Quiz

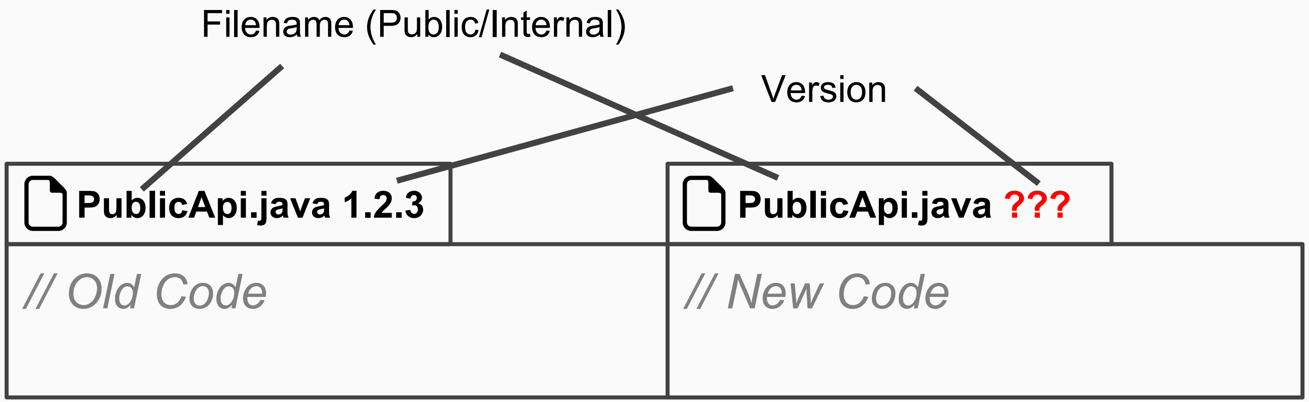
It’s time for:

QUIZ

Semantic Versioning - Quiz - Rules

- 12 questions
- Can be multichoice ;-)
- Time is important
- In most questions you need to answer:

What will be the next version after this change?



Semantic Versioning - Quiz - Link

join.quizizz.com

Game code:

Semantic Versioning - Quiz 1

PublicApi.java 1.2.3

```
public interface PublicApi {

  int calculate(int a, int b);


}
```

PublicApi.java ???

```
public interface PublicApi {

  int calculate(int a, int b);

  double calculate(double a, double b);

}
```

A. 1.2.3

B. 1.2.4

C. 1.3.3

D. 1.3.0

E. 2.2.3

F. 2.0.0

Semantic Versioning - Quiz 1

PublicApi.java 1.2.3

```
public interface PublicApi {

   int calculate(int a, int b);


}
```

PublicApi.java ???

```
public interface PublicApi {

   int calculate(int a, int b);

   double calculate(double a, double b);

}
```

7. Minor version Y (x.Y.z | x > 0) MUST be incremented if new, backwards compatible functionality is introduced to the public API

A. 1.2.3

B. 1.2.4

C. 1.3.3

[✓ D. 1.3.0]

E. 2.2.3

F. 2.0.0

Semantic Versioning - Quiz 2

PublicApi.java 1.2.3

```
public interface PublicApi {

  int calculate(int a, int b);

  double calculate(double a, double b);

}
```

PublicApi.java ???

```
public interface PublicApi {



  double calculate(double a, double b);

}
```

A. 1.2.3

B. 1.2.4

C. 1.3.3

D. 1.3.0

E. 2.2.3

F. 2.0.0

Semantic Versioning - Quiz 2

PublicApi.java 1.2.3

```
public interface PublicApi {

  int calculate(int a, int b);

  double calculate(double a, double b);

}
```

PublicApi.java ???

```
public interface PublicApi {



  double calculate(double a, double b);

}
```

8. Major version X (X.y.z | X > 0) MUST be incremented if any backwards incompatible changes are introduced to the public API

A. 1.2.3

B. 1.2.4

C. 1.3.3

D. 1.3.0

E. 2.2.3

[✓ F. 2.0.0]

Semantic Versioning - Quiz 3

PublicApi.java 1.2.3

```
public class PublicApi {

    public int calculate(int a, int b) {...}


    public double calculate(double a, double b) {...}

}
```

PublicApi.java ???

```
public class PublicApi {

    public int calculate(int a, int b) {...}

    @Deprecated
    public double calculate(double a, double b) {...}

}
```

A. 1.2.3

B. 1.2.4

C. 1.3.3

D. 1.3.0

E. 2.2.3

F. 2.0.0

Semantic Versioning - Quiz 3

PublicApi.java 1.2.3

```
public class PublicApi {

  public int calculate(int a, int b) {...}


  public double calculate(double a, double b) {...}

}
```

PublicApi.java ???

```
public class PublicApi {

  public int calculate(int a, int b) {...}

  @Deprecated
  public double calculate(double a, double b) {...}

}
```

7. Minor version Y (x.Y.z | x > 0) … MUST be incremented if any public API functionality is marked as deprecated.

A. 1.2.3 B. 1.2.4 C. 1.3.3 [✓ D. 1.3.0] E. 2.2.3 F. 2.0.0

Semantic Versioning - Quiz 4

PublicApi.java 1.2.3

```
public class PublicApi {

  public short countDigits(short a) {...}

}
```

PublicApi.java ???

```
public class PublicApi {

  public short countDigits(int a) {...}

}
```

A. 1.2.3

B. 1.2.4

C. 1.3.3

D. 1.3.0

E. 2.2.3

F. 2.0.0

Semantic Versioning - Quiz 4

PublicApi.java 1.2.3

```
public class PublicApi {

  public short countDigits(short a) {...}

}
```

PublicApi.java ???

```
public class PublicApi {

  public short countDigits(int a) {...}

}
```

7. Minor version Y (x.Y.z | x > 0) MUST be incremented if new, backwards compatible functionality is introduced to the public API

A. 1.2.3 | B. 1.2.4 | C. 1.3.3 | [✓ D. 1.3.0] | E. 2.2.3 | F. 2.0.0

Semantic Versioning - Quiz 5

PublicApi.java 1.2.3

```java
public class PublicApi {

  public short calculate(short a, short b) {...}

}
```

PublicApi.java ???

```java
public class PublicApi {

  public int calculate(int a, int b) {...}

}
```

A. 1.2.3

B. 1.2.4

C. 1.3.3

D. 1.3.0

E. 2.2.3

F. 2.0.0

Semantic Versioning - Quiz 5

PublicApi.java 1.2.3

```
public class PublicApi {

  public short calculate(short a, short b) {...}

}
```

PublicApi.java ???

```
public class PublicApi {

  public int calculate(int a, int b) {...}

}
```

8. Major version X (X.y.z | X > 0) MUST be incremented if any backwards incompatible changes are introduced to the public API. (returned int instead of short)

A. 1.2.3

B. 1.2.4

C. 1.3.3

D. 1.3.0

E. 2.2.3

[✓ F. 2.0.0]

Semantic Versioning - Quiz 6

***.public.AbstractShape.java 1.2.3**

```
public abstract class AbstractShape {



}
```

***.public.AbstractShape.java ???**

```
public abstract class AbstractShape {

  public double calculateArea() {...}

}
```

A. 1.2.3

B. 1.2.4

C. 1.3.3

D. 1.3.0

E. 2.2.3

F. 2.0.0

Semantic Versioning - Quiz 6

*.public.AbstractShape.java 1.2.3

```
public abstract class AbstractShape {



}
```

*.public.AbstractShape.java ???

```
public abstract class AbstractShape {

   public double calculateArea() {...}

}
```

7. Minor version Y (x.Y.z | x > 0) MUST be incremented if new, backwards compatible functionality is introduced to the public API.

A. 1.2.3

B. 1.2.4

C. 1.3.3

[✓ D. 1.3.0]

E. 2.2.3

F. 2.0.0

Semantic Versioning - Quiz 7

*.public.AbstractShape.java 1.2.3

```
public abstract class AbstractShape {

   public double calculateArea() {...}


}
```

*.public.AbstractShape.java ???

```
public abstract class AbstractShape {

   public double calculateArea() {...}

   public abstract double setP(double a, double b);

}
```

A. 1.2.3

B. 1.2.4

C. 1.3.3

D. 1.3.0

E. 2.2.3

F. 2.0.0

Semantic Versioning - Quiz 7

***.public.AbstractShape.java 1.2.3**

```
public abstract class AbstractShape {

   public double calculateArea() {...}


}
```

***.public.AbstractShape.java ???**

```
public abstract class AbstractShape {

   public double calculateArea() {...}

   public abstract double setP(double a, double b);

}
```

8. Major version X (X.y.z | X > 0) MUST be incremented if any backwards incompatible changes are introduced to the public API.

A. 1.2.3

B. 1.2.4

C. 1.3.3

D. 1.3.0

E. 2.2.3

[✓ F. 2.0.0]

Semantic Versioning - Quiz 8

*.public.AbstractShape.java 1.2.3

```
public abstract class AbstractShape {

  public double calculateArea() {...}

  public abstract double setP(double a, double b);


}
```

*.public.AbstractShape.java ???

```
public abstract class AbstractShape {

  public double calculateArea() {...}

  public abstract double setP(double a, double b);

  protected abstract double setP(int a, int b);
}
```

A. 1.2.3

B. 1.2.4

C. 1.3.3

D. 1.3.0

E. 2.2.3

F. 2.0.0

Semantic Versioning - Quiz 8

*.public.AbstractShape.java 1.2.3

```
p

   public abstract double setP(double a, double b);


}
```

*.public.AbstractShape.java ???

```
   public abstract double setP(double a, double b);

   protected abstract double setP(int a, int b);
}
```

8. Major version X (X.y.z | X > 0) MUST be incremented if any backwards incompatible changes are introduced to the public API.

A. 1.2.3

B. 1.2.4

C. 1.3.3

D. 1.3.0

E. 2.2.3

[✓ F. 2.0.0]

Semantic Versioning - Quiz 9

PublicApi.java 1.2.3

```java
public interface PublicApi {

   int open(String file);

}
```

PublicApi.java ???

```java
public interface PublicApi {

   int open(String file) throws IOException;

}
```

A. 1.2.3

B. 1.2.4

C. 1.3.3

D. 1.3.0

E. 2.2.3

F. 2.0.0

Semantic Versioning - Quiz 9

PublicApi.java 1.2.3

```java
public interface PublicApi {

  int open(String file);


}
```

PublicApi.java ???

```java
public interface PublicApi {

  int open(String file) throws IOException;


}
```

8. Major version X (X.y.z | X > 0) MUST be incremented if any backwards incompatible changes are introduced to the public API.

A. 1.2.3

B. 1.2.4

C. 1.3.3

D. 1.3.0

E. 2.2.3

[✓ F. 2.0.0]

Semantic Versioning - Quiz 10

PublicApi.java 1.2.3

```
public class PublicApi {

  public short calculate(short a, short b) {
      throw new IllegalArgumentException();
  }

}
```

PublicApi.java ???

```
public class PublicApi {

  public short calculate(short a, short b) {
      return 0;
  }

}
```

A. 1.2.3

B. 1.2.4

C. 1.3.3

D. 1.3.0

E. 2.2.3

F. 2.0.0

Semantic Versioning - Quiz 10

PublicApi.java 1.2.3

```
public class PublicApi {

  public short calculate(short a, short b) {
      throw new IllegalArgumentException();
  }

}
```

PublicApi.java ???

```
public class PublicApi {

  public short calculate(short a, short b) {
      return 0;
  }

}
```

6. Patch version Z (x.y.Z | x > 0) MUST be incremented if only backwards compatible bug fixes are introduced. A bug fix is defined as an internal change that fixes incorrect behavior.

A. 1.2.3

[✓ B. 1.2.4]

C. 1.3.3

D. 1.3.0

E. 2.2.3

F. 2.0.0

Semantic Versioning - Quiz 11

PublicApi.java 1.2.3

```java
public class PublicApi {

   public int inverse(int a) {



      return -a;
   }

}
```

PublicApi.java ???

```java
public class PublicApi {

   public int inverse(int a) {
      if (a == Integer.MIN_VALUE) {
         throw new IllegalArgumentException();
      }
      return -a;
   }

}
```

A. 1.2.3

B. 1.2.4

C. 1.3.3

D. 1.3.0

E. 2.2.3

F. 2.0.0

Semantic Versioning - Quiz 11

PublicApi.java 1.2.3

```
public class PublicApi {

  public int inverse(int a) {



      return -a;



}
```

PublicApi.java ???

```
public class PublicApi {

  public int inverse(int a) {
      if (a == Integer.MIN_VALUE) {
          throw new IllegalArgumentException();
      }
      return -a;
```

6. Patch version Z (x.y.Z | x > 0) MUST be incremented if only backwards compatible bug fixes are introduced. A bug fix is defined as an internal change that fixes incorrect behavior.

A. 1.2.3 [✓ B. 1.2.4] C. 1.3.3 D. 1.3.0 E. 2.2.3 F. 2.0.0

Semantic Versioning - Quiz 12

InternalApi.java 1.2.3

```
class InternalApi {
  public int max(int x, int y) { … }


}
```

InternalApi.java ???

```
class InternalApi {
  public int max(int x, int y) { … }
  public int min(int x, int y) { … }
  public int avg(int x, int y) { … }
  public double sin(double x) { … }
}
```

A. 1.2.3

B. 1.2.4

C. 1.3.3

D. 1.3.0

E. 2.2.3

F. 2.0.0

Semantic Versioning - Quiz 12

InternalApi.java 1.2.3

```java
class InternalApi {
  public int max(int x, int y) { … }
}
```

InternalApi.java ???

```java
class InternalApi {
  public int max(int x, int y) { … }
  public int min(int x, int y) { … }
  public int avg(int x, int y) { … }
  public double sin(double x) { … }
}
```

7. Minor version Y (x.Y.z | x > 0) … MAY be incremented if substantial new functionality or improvements are introduced within the private code... Patch version MUST be reset to 0 when minor version is incremented.

A. 1.2.3 [✓ B. 1.2.4] C. 1.3.3 [✓ D. 1.3.0] E. 2.2.3 F. 2.0.0

Semantic Versioning - Quiz 13

Please select ONE correct precedence

1.0.0-alpha < 1.0.0-alpha.1 < 1.0.0-alpha.beta < 1.0.0-beta < 1.0.0-beta.11 < 1.0.0-beta.2 < 1.0.0-rc.1 < 1.0.0

1.0.0-alpha.1 < 1.0.0-alpha < 1.0.0-alpha.beta < 1.0.0-beta < 1.0.0-beta.2 < 1.0.0-beta.11 < 1.0.0-rc.1 < 1.0.0

1.0.0-alpha < 1.0.0-alpha.1 < 1.0.0-alpha.beta < 1.0.0-beta < 1.0.0-beta.2 < 1.0.0-beta.11 < 1.0.0-rc.1 < 1.0.0

1.0.0-alpha < 1.0.0-alpha.beta < 1.0.0-alpha.1 < 1.0.0-beta < 1.0.0-beta.2 < 1.0.0-beta.11 < 1.0.0-rc.1 < 1.0.0

Semantic Versioning - Quiz 13

11. … Precedence MUST be calculated by separating the version into major, minor, patch and pre-release identifiers in that order (Build metadata does not figure into precedence)... When major, minor, and patch are equal, a pre-release version has lower precedence than a normal version. Precedence for two pre-release versions ... MUST be determined by comparing each dot separated identifier from left to right until a difference is found as follows: identifiers consisting of only digits are compared numerically and identifiers with letters or hyphens are compared lexically in ASCII sort order. Numeric identifiers always have lower precedence than non-numeric identifiers.

1.0.0-alpha < 1.0.0-alpha.1 < 1.0.0-alpha.beta < 1.0.0-beta < 1.0.0-beta.2 < 1.0.0-beta.11 < 1.0.0-rc.1 < 1.0.0

1.0.0-alpha < 1.0.0-alpha.beta < 1.0.0-alpha.1 < 1.0.0-beta < 1.0.0-beta.2 < 1.0.0-beta.11 < 1.0.0-rc.1 < 1.0.0

Outside competition

Outside competition

*.public.AbstractShape.java 1.2.3

```java
public abstract class AbstractShape {


}
```

*.public.AbstractShape.java ???

```java
public abstract class AbstractShape {

   abstract double setP(short a, short b) {...}

}
```

A. 1.2.3

B. 1.2.4

C. 1.3.3

D. 1.3.0

E. 2.2.3

F. 2.0.0

Outside competition

*.public.AbstractShape.java 1.2.3

```
public abstract class AbstractShape {


}
```

*.public.AbstractShape.java ???

```
public abstract class AbstractShape {

   abstract double setP(short a, short b) {...}

}
```

Is it breaking change or not?

A. 1.2.3

B. 1.2.4

C. 1.3.3

D. 1.3.0

E. 2.2.3

F. 2.0.0

Outside competition

*.public.AbstractShape.java 1.2.3

```
public abstract class AbstractShape {


}
```

*.public.AbstractShape.java ???

```
public abstract class AbstractShape {

  abstract double setP(short a, short b);

}
```

Is it breaking change or not?
It depends on how this class is used.
All child classes (in package/library) need to override it.
But how classes outside package can override it?

A. 1.2.3

B. 1.2.4

C. 1.3.3

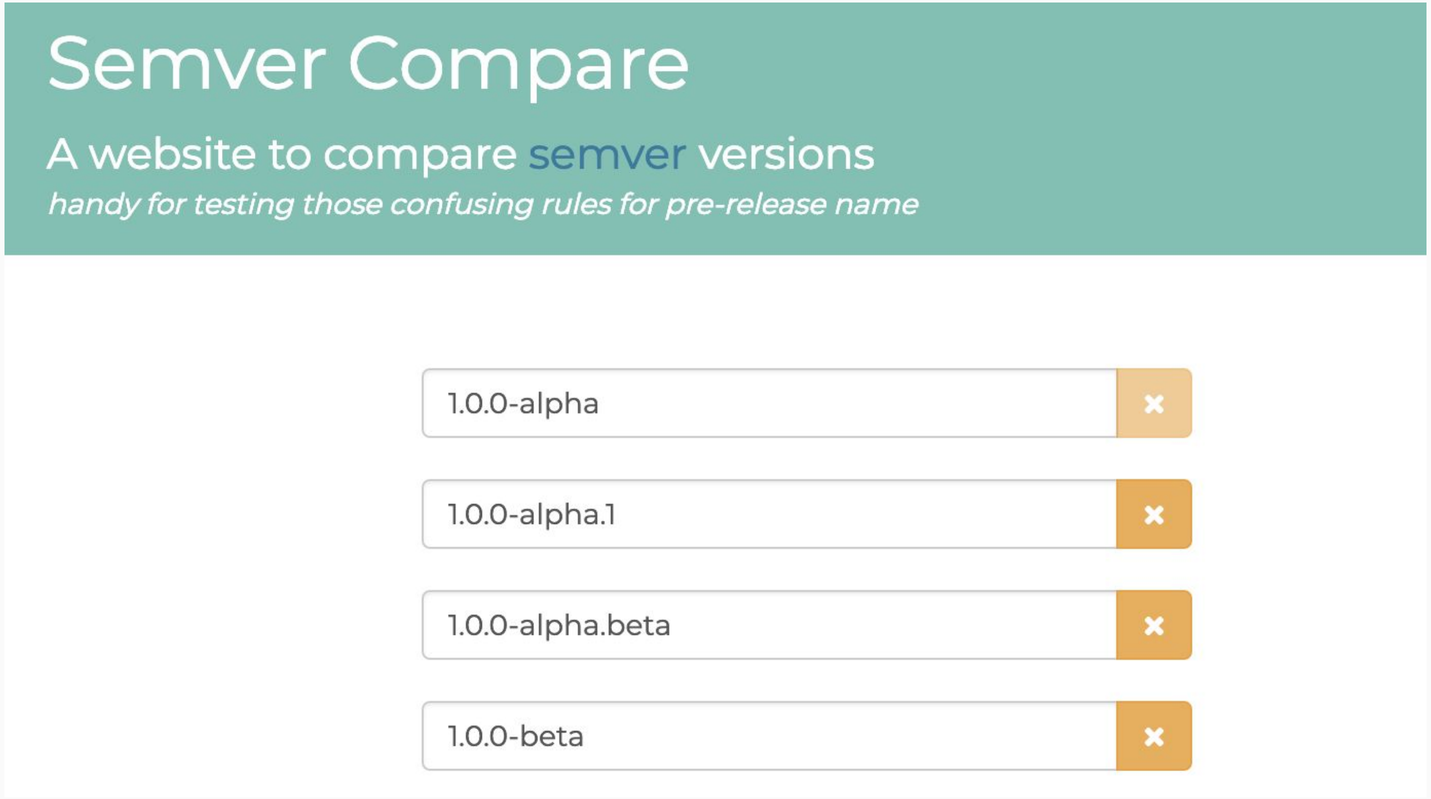
D. 1.3.0

E. 2.2.3

F. 2.0.0

Sem Ver Compare online

http://semvercompare.azurewebsites.net



Semver Compare

A website to compare semver versions

handy for testing those confusing rules for pre-release name

1.0.0-alpha

1.0.0-alpha.1

1.0.0-alpha.beta

1.0.0-beta