serialVersionUID について 2021/06/21

1 Java 環境の確認

コマンドプロンプトにて、Java が起動できるか確認します。

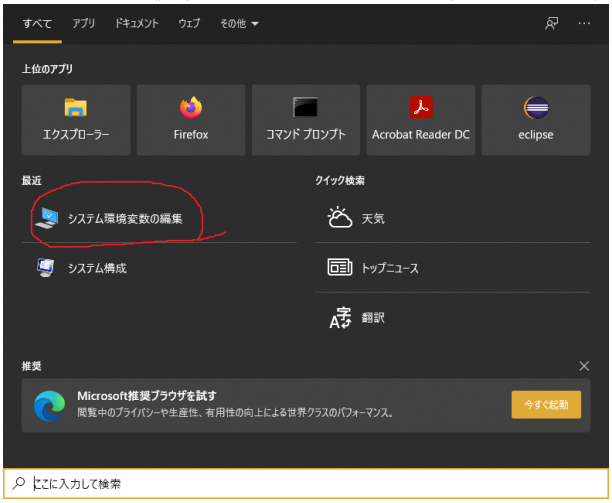
```
> java -version
openjdk version "11" 2018-09-25
OpenJDK Runtime Environment AdoptOpenJDK (build 11+28-201810022317)
OpenJDK 64-Bit Server VM AdoptOpenJDK (build 11+28-201810022317, mixed mode)
```

同様に Javac も確認します。

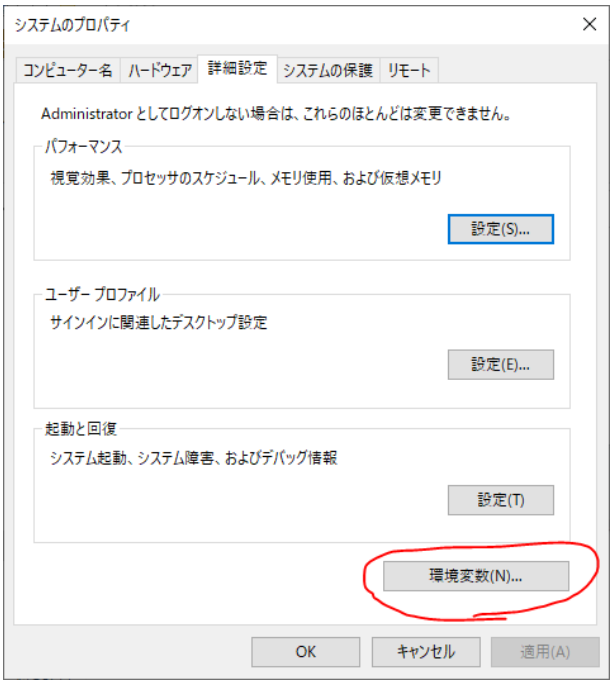
```
> javac -version
javac 11
```

もしも、起動できなかったら、以下の作業をします。問題なく起動できた場合は、次のセクションに進んでください。

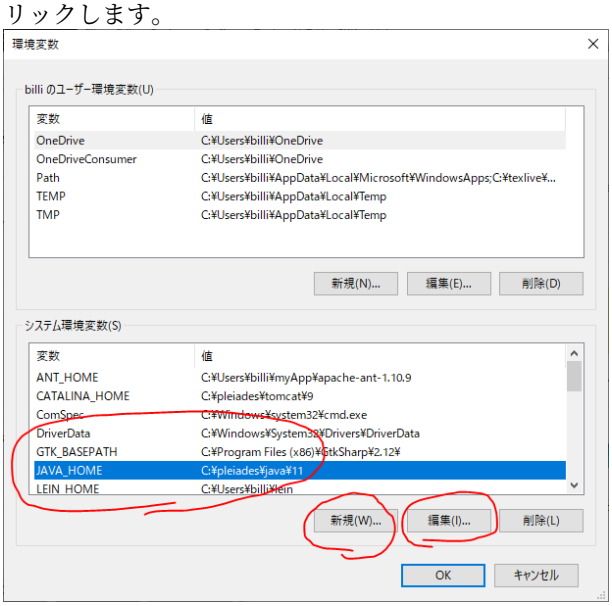
1. タスクバーの検索アイコンをクリックし、「システム環境変数の編集」と入力します。



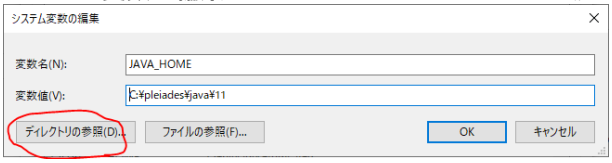
2. 開かれたウィンドウの **システム環境変数の編集**を選択します。



3. 環境変数ダイアログが開くので、下の **システム環境変数** の欄を確認します。JAVA_HOME が無ければ **新規**で作成します。JAVA_HOME の値を変更する必要があるかもしれません。その場合は **編集** をクリックします。



4. **システム変数の編集** ダイアログで JAVA_HOME の値を設定します。



ここでは、pleiades フォルダの中の java フォルダの 11 を指定しています。Java11 を使っています。これは各自の環境で読み換えてください。

システム環境変数の設定を変更した場合、それを有効にするには、コマンドプロンプトを閉じて、もう一度開きなおしてください。そののち、`java -version` と `javac -version` を確認してみてください。

2 serialVersionUID の働きについて

2.1 サンプル・コード

以下のコードをサンプルにします。適当な場所にフォルダを作成し、お気に入りのエディタで以下のコードを入力し、Fruit.java というファイル名で保存してください。

リスト 1　Fruit.java

```
1 import java.io.Serializable;
2
3 public class Fruit implements Serializable {
4
5     private String name;
6     private int price;
7
8 }
```

2.2 コンパイルして serialVersionUID を確認する

Fruit.java があるフォルダで コマンドプロンプトを開き、[*1]コンパイルします。

```
> javac *.java
```

Fruit.class ができているので、以下のコマンドを実行します。

```
> serialver Fruit
```

すると、以下のような文字列が出力されます。(数字列は各環境で違います)

```
Fruit:    private static final long serialVersionUID = 7481014788150016266L;
```

2.3 クラスの記述を変更する

クラスの記述を少し変更してみます。以下のようにしてみます。

リスト 2　Fruit.java

```
1 import java.io.Serializable;
2
3 public class Fruit implements Serializable {
4
5     private final String name = "Apple";
```

[*1] ファイル・エクスプローラでそのフォルダを開き、ファイル・エクスプローラのアドレス欄に cmd と入力し、Enter キーを押下すると、そのフォルダでコマンドプロンプトを開くことができます。

```
6       private final int price = 200;
7
8   }
```

フィールド変数を final にしてみました。初期化しろと言われるので、初期値を入れてます。これでコンパイルしてみます。

```
> javac *.java
> serialver Fruit
```

すると、以下のような出力になります。

```
Fruit:    private static final long serialVersionUID = 2521861718812792199L;
```

変更を元に戻して、コンパイルしなおして実行してみますと、元の数字列になっているのが確認できます。
いろいろな箇所を変更して試してみてください。

2.4 フィールドやメソッドを変更すると、UID が変わる

フィールドの型を変更したり、変数名を変更したり、メソッドを追加したりすると、UID の数字列が変わるのが確認できたはずです。
UID が変わるということは、JVM が、そのインスタンスは別物だと認識しているということになります。

2.5 serialVersionUID を設定すると

クラスの記述の中に、serialVersionUID を指定すると、クラス定義を変更しても、JVM に同一のインスタンスであると認識してもらうことができます。
仮に、以下のように記述してみます。

リスト 3　Fruit.java

```
1   import java.io.Serializable;
2
3   public class Fruit implements Serializable {
4
5       private static final long serialVersionUID = 7481014788150016266L;
6       private String name = "Apple";
7       private int price = 200;
8
9   }
```

これで、コンパイルした後、UID を確認します。

```
> javac *.java
> serialver Fruit
Fruit:    private static final long serialVersionUID = 7481014788150016266L;
```

上記のクラス定義を少し変更します。

リスト 4　Fruit.java

```
1  import java.io.Serializable;
2
3  public class Fruit implements Serializable {
4
5      private static final long serialVersionUID = 7481014788150016266L;
6      private final String name = "Apple";
7      private final int price = 200;
8
9  }
```

これでコンパイルし、UID を確認すると、同じであることがわかります。

```
> javac *.java
> serialver Fruit
Fruit:    private static final long serialVersionUID = 7481014788150016266L;
```

2.6　結論

serialVersionUID を指定するということは、生成したインスタンスがもし変更されていたとしても、同じインスタンスであると JVM に認識させるということになります。

3　ファイルの読み書きで試してみる

3.1　シリアライズが必要な処理とは？

シリアライズとは「直列化」と訳されます。Java のデータを他の Java プログラムとやりとりしたいときに、シリアライズされて相手プログラムに渡されます。たとえば、ファイルのやりとりの場合は、シリアライズされてファイルとして保存されます。そして、別のプログラムで読込み、デシリアライズされます。

ここでは、実際にファイルのやりとりをしてみます。

3.2　サンプルコード

以下のような Human クラスを例にとって考えます。[*2]

リスト 5　Human.java

```
1  import java.io.Serializable;
2
3  public class Human implements Serializable {
4
5      private String name;
6      private int age;
7
8      public Human( String name, int age ) {
9          this.name = name;
```

[*2] 出典：「serialVersionUID って何なの？書くのめんどい」http://java-study.blog.jp/archives/1021151333.html

```
10          this.age = age;
11      }
12
13      public String getName() {
14          return name;
15      }
16
17      public int getAge() {
18          return age;
19      }
20  }
```

以下が、Human クラスをシリアライズ化してファイルに書き込む処理を記述したコードです。

リスト 6　SerializableWrite.java

```
1  import java.io.FileOutputStream;
2  import java.io.ObjectOutputStream;
3
4  public class SerializableWrite {
5      public static void main (String[] args) throws Exception {
6          Human human = new Human("John", 15);            // <1>
7
8          // オブジェクトをファイルに書き込む
9          try (FileOutputStream fs = new FileOutputStream("human.obj");
10              ObjectOutputStream oos = new ObjectOutputStream (fs)) {
11             oos.writeObject( human );                    // <2>
12         }
13     }
14 }
```

<1> --　Human クラスのインスタンスを作成しています。

<2> --　インスタンスをシリアライズ化して human.obj という名前で書き込んでいます。

そして、シリアライズ化してあるファイルを読み込んで、デシリアライズする処理です。

リスト 7　MainSerializeRead.java

```
1  import java.io.FileInputStream;
2  import java.io.ObjectInputStream;
3
4  public class MainSerializeRead {
5      public static void main (String[] args) throws Exception {
6
7          // オブジェクトをファイルから読み込む
8          try (FileInputStream fs = new FileInputStream("human.obj");
9              ObjectInputStream oos = new ObjectInputStream (fs)) {
10
11             Human human = (Human)oos.readObject();          // <1>
12             System.out.println( human.getName() + ":" + human.getAge());
13         }
14     }
15 }
```

<1> --　human.obj を読み込んで デシリアライズし、Human 型にキャストしています。

3.3 コンパイルして実行

コンパイルします。

```
> javac *.java
```

実行します。まず、書き込む処理 (SerializableWrite) からです。

```
> java SerializableWrite
```

すると、このフォルダに human.obj ができています。これは、Human クラスのインスタンスをシリアライズして保存したものです。

読み込んでみます。

```
> java MainSerializeRead
Johe:15
```

読み込めています。

3.4 Human クラスを変更してみる

ここで、Human クラスを変更してみます。フィールドの変数を final にしてみます。

リスト 8　Human.java

```
1  import java.io.Serializable;
2
3  public class Human implements Serializable {
4
5      private final String name;
6      private final int age;
7
8      public Human( String name, int age ) {
9          this.name = name;
10         this.age = age;
11     }
12
13     public String getName() {
14         return name;
15     }
16
17     public int getAge() {
18         return age;
19     }
20 }
```

Human クラスだけ コンパイルします。

```
> javac Human.java
```

これで human.obj の読込みをしてみます。

```
> java MainSerializeRead
Exception in thread “main” java.io.InvalidClassException: Human; local class incompatible: stream classdesc serialVersionUID = -4742490797832013657, local class serialVersionUID = -6199831495933285688
```

serialVersionUID が違うため、クラスが不正であるという例外が発生しています。

3.5 serialVersionUID を指定してやってみる

今度は、Human クラスに serialVersionUID を指定してやってみます。

リスト 9 Human.java

```
1  import java.io.Serializable;
2
3  public class Human implements Serializable {
4
5      private static final long serialVersionUID = 1L;
6      private String name;
7      private int age;
8
9      public Human( String name, int age ) {
10         this.name = name;
11         this.age = age;
12     }
13
14     public String getName() {
15         return name;
16     }
17
18     public int getAge() {
19         return age;
20     }
21 }
```

コンパイルして、シリアライズして保存します。

```
> javac Human.java
> java SerializableWrite
```

Human クラスを変更します。今度も フィールド変数を final にします。

リスト 10 Human.java

```
1  import java.io.Serializable;
2
3  public class Human implements Serializable {
4
5      private static final long serialVersionUID = 1L;
6      private final String name;
7      private final int age;
```

```
8
9       public Human( String name, int age ) {
10          this.name = name;
11          this.age = age;
12      }
13
14      public String getName() {
15          return name;
16      }
17
18      public int getAge() {
19          return age;
20      }
21  }
```

コンパイルして、デシリアライズ、読込みをします。

```
> javac Human.java
> java MainSerializeRead
Jone:15
```

読込みに成功しています。

4 つまり

serialVersionUID を指定しておくと、クラスをインスタンス化して、シリアライズしたあと、そのクラスの定義を変更したとしても、デシリアライズができるということになります。

シリアライズは、Java のオブジェクト (インスタンス) を ファイルやネットワークに渡すときに使われます。そのオブジェクトを受け取るのが Java のプログラムである場合です。シリアライズは「直列化」と訳されます。