Differences between Java 7's generics and Java 10 (Projet Valhalla) ones

1. ContainsAny

Experiences

I am going to compile a classfile the simplest possible in order to determine the indicators of an any variable. To do so, I am going to work on these differents cases :

1. Simplest class parameterized by any type
2. Static method parameterized by any type
3. Static method parameterized by 2 any types
4. Nested class parameterized by any type
5. Nested class using enclosing any type
6. Sub class parameterizing the super class
7. Build a class with ASM, resulting of the transformation of an avar parameterized class

1.1. Simplest class parameterized by any type

1.1.1. Observations

This simple class is parameterized by an avar T. It has one field T t, the corresponding getter and setter and a constructor initializing the field t. A witness class is also compiled in order to compare the results after compiling these two classes.

```
public class Class1<any T> {
    T v;
    public Class1(T v) { this.v = v; }
    public T getV() { return v; }
}
```

```
public class Class1Witness<T> {
    T v;
    public Class1Witness(T v) { this.v = v; }
    public T getV() { return v; }
}
```

We observe the creation of an empty inner interface : **Class1$$any.class**. The class Class1 implements it.

Also, we observe the apparition of the structure **TypeVariablesMap** providing informations on type variables. In the Class1 class, T has the FLAG **[ANY]**, but in the witness class, it has **[]**.

Appearance of the bytecode instruction **TYPED** before every instructions concerning an avar.
Appearance of the attribute **TypeVar** :

- In argument of each instruction **TYPED**

- In attribut of the instruction **descriptor** for the field t, when it is directly the bound inside Class1Witness
    - Each **TypeVar** contains the bound `Object` and not "_" meaning "This type is erased".

1.2. Static method parameterized by any type

1.2.1. Observations

The names of the compiled classes are Class2 and Class2Witness.
After compiling Class2, we first observe the creation of a static inner class where the name is composed like the following :
EnclosingClass + $ + StaticMethodName + $ + CodeSequence
In our case, compiling the static method **methodStatic2** of signature `public static <any T> void methodStatic2(T)` inside the class **Class2** creates the synthetic class **Class2$methodStatic2$1601768860**.
Any code can be put inside the static method, it will always produce the same bytecode instructions and move the implementation inside the inner class created.
The bytecode instructions are :

```
public static <T extends java.lang.Object> void methodStatic2(T);
    descriptor: (Ljava/lang/Object;)V
    flags: ACC_PUBLIC, ACC_STATIC
    Code:
      stack=2, locals=1, args_size=1
         0: new           #10                 // class fr/upem/any/Class2$methodStatic2$1601768860
         3: dup
         4: invokespecial #12                 // Method fr/upem/any/Class2$methodStatic2$1601768860."<init>":()V
         7: aload_0
         8: invokevirtual #16                 // Method fr/upem/any/Class2$methodStatic2$1601768860.methodStatic2:(Ljava/lang/Object;)V
        11: return
      LineNumberTable:
        line 1: 0
    Signature: #20                          // <T:Ljava/lang/Object;>(TT;)V
    TypeVariablesMap:
      Lfr/upem/any/Class2;::methodStatic2(Ljava/lang/Object;)V:
        Tvar  Flags  Bound
        T     [ANY]  Ljava/lang/Object;
```

QUESTION : Is the argument boxed inside an Object reference ?
Presence of the TypeVariablesMap with a T having the FLAG : [ANY].
No TypeVar or erased token.
Also, it seems that is append to the constructor name :

```
public fr.upem.any.Class2$methodStatic2$1601768860<T>();
    descriptor: ()V
    flags: ACC_PUBLIC
    Code:
      stack=1, locals=1, args_size=1
         0: aload_0
         1: invokespecial #6                  // Method java/lang/Object."<init>":()V
         4: return
      LineNumberTable:
        line 1: 0
```

Moreover, the is also append to the class name in the header.

```
public class fr.upem.any.Class2$methodStatic2$1601768860<T> <T extends java.lang.Object> extends java.lang.Object
        this: #46                                       // "fr/upem/any/Class2$methodStatic2$1601768860<T>"
```

But this suffixe does not exist in the name file nor in the inner classes declarations :

```
InnerClasses:
public static #42= #41 of #39;          // methodStatic2$1601768860=class fr/upem/any/Class2$methodStatic2$1601768860 of class fr/upem/any/Class2
```

1.2.2. Content of the synthetized static inner class Class2$methodStatic2$1601768860

This static inner class contains the code of the static avars parameterized method.
Presence of a bytecode instruction TYPED just before the loading of the firts - avar - argument.
Its load is an aload/_1 as opposed to the witness version where it is aload/_0 because the execution is inside a non static method.
We have a TypeVariablesMap with a T having the FLAG : ANY.
A bootstrap method is used to perform T.toString() :

```
        ...
        7: invokedynamic #31,  0             // InvokeDynamic #0:toString:(T)Ljava/lang/String;
        12: invokevirtual #37                // Method java/io/PrintStream.println:(Ljava/lang/String;)V
        ...
        ...
        ...
BootstrapMethods:
  0: #26 invokestatic java/lang/invoke/ObjectibleDispatch.metafactory:(Ljava/lang/invoke/MethodHandles$Lookup;Ljava/lang/String;Ljava/lang/invoke/MethodType;[Ljava/lang/Object;)Ljava/lang/invoke/CallSite;
    Method arguments:
```

Instead of calling Bounds#toString(). It allows the dynamic detection of the right equals/hashcode/toString method.
The last additional information is the location of 2 inner classes used :

```
InnerClasses:
  public static final #22= #21 of #19;    // Lookup=class java/lang/invoke/MethodHandles$Lookup of class java/lang/invoke/MethodHandles
  public static #42= #41 of #39;          // methodStatic2$1601768860=class fr/upem/any/Class2$methodStatic2$1601768860 of class fr/upem/any/Class2
```

1.3. Static method parameterized by 2 any types U and V

1.3.1. Observations

```
public class Class3 {
    public static <any T, any V> void methodStatic3(T t, V v) {
        System.out.println(t.toString() + v.toString());
    }
}
```

For a class named Class3, parameterized by any U and any V : Same observations, an inner class **Class3$methodStatic3$763128888** is created.
2 Entries for the TypeVariablesMap.
Presence of TypeVar structures following the same criteria.

1.3.2. Content of the synthetized static inner class Class3$methodStatic3$763128888

3 Invokedynamic are used :

```
  public void methodStatic3(T, V);
    descriptor: (TT;TV;)V
    flags: ACC_PUBLIC
    Code:
      stack=3, locals=3, args_size=3
         0: getstatic     #12                 // Field java/lang/System.out:Ljava/io/PrintStream;
         3: typed         #14                 // TypeVar T/Ljava/lang/Object
         6: aload_1
         7: invokedynamic #31,  0             // InvokeDynamic #0:toString:(T)Ljava/lang/String;
        12: typed         #32                 // TypeVar V/Ljava/lang/Object
        15: aload_2
        16: invokedynamic #35,  0             // InvokeDynamic #0:toString:(V)Ljava/lang/String;
        21: invokedynamic #47,  0             // InvokeDynamic #1:makeConcatWithConstants:(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;
        26: invokevirtual #53                 // Method java/io/PrintStream.println:(Ljava/lang/String;)V
        29: return
      LineNumberTable:
        line 8: 0
        line 9: 29
    Signature: #70                          // (TT;TV;)V
```

Question : Where are passed the argument to the bootstrap methods ?? :S (Especially for makeConcatWithConstants

1.4. Nested class parameterized by any type

1.4.1. Observations

Obvious reaction : Manipulatin the inner class like a normal one and creating the $$any interface on it.

1.5. Nested class using enclosing any type

1.5.1. Observations

From a class named Class5.java, creates 4 classfiles.
The Class5.class itself and its $any interface Class5$$any.class. The inner class Class5$Classe5Inner.class and its $any interface Class5$Classe5Inner$$any.class.
No specific carryforward of the code in another class as done in the case of the static method export.
Presence of types inside the TypeVariablesMap of Class5.class :

- **Lfr/upem/any/Class5$Classe5Inner;:**
- **Lfr/upem/any/Class5;:** without any **FLAGS** or **BOUNDS**.
  TYPED instructions and TypeVar attributes are also present.
  The TypeVar is transfered/copied to the inner classfile.
  The inner class **Class5$Classe5Inner** and its interface Class5$Classe5Inner$$any are also noted :

```
InnerClasses:
public #5= #4 of #2;           // Classe5Inner=class fr/upem/any/Class5$Classe5Inner of class fr/upem/any/Class5
public static #27= #26 of #4;  // $any=class fr/upem/any/Class5$Classe5Inner$$any of class fr/upem/any/Class5$Classe5Inner
```

Question : Why is there the following difference between the names of the constructors ?

```
            public fr.upem.any.Class5$Classe5Inner<>(T);  // Contains some avars
            public fr.upem.any.Class5Witness$Classe5Inner(T); // Contains no avars
```

The first class is public class fr.upem.any.Class5$Classe5Inner<> (notice the '<>') and the witness is public class fr.upem.any.Class5Witness$Classe5Inner.

1.6. Sub class parameterized by any type parameterizing the super class

1.6.1. Observation

From a class named Class6.java and its subclass Class6Child.java, creates 4 classfiles.
The 2 classes are treated like any classes parameterized by an any variable type. The inner interface **Class6$$any** is created for Class6
and the inner interface Class6Child$$any for Class6Child.

```
TypeVariablesMap:
  Lfr/upem/any/Class6;:
    Tvar  Flags  Bound
    T     [ANY]  Ljava/lang/Object;
TypeVariablesMap:
  Lfr/upem/any/Class6Child;:
    Tvar  Flags  Bound
    T     [ANY]  Ljava/lang/Object;
```

2.X. Translation rules deducted

2.X.Y

We will also have to "clean" classes not using specialization but still using JAVA 10 structures like TypeVariablesMap.

Pending

- ArrayType (already done inside the code)