FACULTY OF INFORMATICS

Advanced java programming

Assignment 4

Author FEDERICO LAGRASTA

CLAUDIO MAGGIONI

December 11, 2022

Exercise A: Concurrency

As no synchronization between the started threads exists, all the possible permutations of "tX" where *x* is 1, 2 or 3 are valid outputs. Namely, all possible combinations are:

- "t1 t2 t3 "
- "t1 t2 t3 "
- "t2 t1 t3 "
- "t2 t3 t1 "
- "t3 t1 t2 "
- "t3 t2 t1 "

Exercise B: Concurrency

1. No, as the method is `synchronized` and requires a mutually exclusive lock to be acquired to execute it, therefore only one thread at a time can execute;
2. Yes, as *getCounter()* is not marked with the `synchronized` keyword;
3. Yes, as one thread may acquire the lock for *incrementCounter()* while the other just released it and is executing *getCounter()*;
4. Yes, as the updates to the *count* variable in *Counter* are atomic and safe, therefore the count will always be exact;
5. The object will acquire and release the intrinsic lock of the *counter* object defined at line 17.

Exercise C: Concurrency

Yes, as the foreach loop may indeed run simultaneously in multiple threads. According to the Java 17 standard library documentation for `Collections.synchronizedList(...)`[1], in order to obtain mutually exclusive access in the iteration we would need to use the intrinsic lock of the list object wrapping the foreach loop with a `synchronized (integerList)` block.

Exercise D: JVM Architecture

1. As invoking `javap Secret` reports:

   ```
   Warning: File ./Secret.class does not contain class Secret
   Compiled from ''Login.java''
   ```

   the name of the original Java file is `Login.java`;

2. According to the output of `javap -c -l Secret`, specifically these lines:

   ```
    0: ldc #55  // String Hello, try to find the username and the password [...]
   71: ldc #79  // String Congratulations, now write the username and the [...]
   [...]
   LineNumberTable:
         line 27: 0
   [...]
         line 41: 68
         line 42: 76
   [...]
   ```

[1] `https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Collections.html#synchronizedList(java.util.List)`

the string starting with “Hello” is stored in tag 55 of the constant pool and is loaded at line 27. The string starting with “Congratulations” is stored in tag 79 of the constant pool and is loaded at line 42.

3. According to the output of `javap Secret` showing the class’s interface:

   ```
   public class Login {
     public Login();
     public static char transform(char);
     public static boolean access(java.lang.String, java.lang.String);
     public static void main(java.lang.String...);
   }
   ```

   there are three `public static` methods in the class.

To aid the explanation of how we found the secret password we report the full output of *javap -c -l Secret*:

```
public class Login {
  public Login();
    Code:
       0: aload_0
       1: invokespecial #1        // Method java/lang/Object."<init>":()V
       4: return

  public static char transform(char);
    Code:
       0: iload_0
       1: bipush        97
       3: isub
       4: istore_1
       5: iload_1
       6: iconst_1
       7: isub
       8: bipush        26
      10: irem
      11: bipush        97
      13: iadd
      14: istore_2
      15: iload_2
      16: i2c
      17: ireturn

  public static boolean access(java.lang.String, java.lang.String);
    Code:
       0: ldc           #7        // String tvvvvqfstuspohqbttxpse
       2: astore_2
       3: ldc           #9        // String
       5: astore_3
       6: iconst_0
       7: istore        4
       9: iload         4
      11: aload_2
      12: invokevirtual #11       // Method java/lang/String.length:()I
      15: if_icmpge     40
      18: aload_3
      19: aload_2
      20: iload         4
      22: invokevirtual #17       // Method java/lang/String.charAt:(I)C
      25: invokestatic  #21       // Method transform:(C)C
```

```
    28: invokedynamic #27,  0  // InvokeDynamic #0:makeConcatWithConstants:
                               // (Ljava/lang/String;)Ljava/lang/String;
    33: astore_3
    34: iinc          4, 1
    37: goto          9
    40: aload_0
    41: ldc           #31      // String admin
    43: invokevirtual #33      // Method java/lang/String.equals:(Ljava/lang/Object;)Z
    46: ifeq          70
    49: aload_1
    50: aload_3
    51: invokevirtual #37      // Method java/lang/String.toString:()Ljava/lang/String;
    54: invokevirtual #33      // Method java/lang/String.equals:(Ljava/lang/Object;)Z
    57: ifeq          70
    60: getstatic     #41      // Field java/lang/System.out:Ljava/io/PrintStream;
    63: ldc           #47      // String Success!! Access granted
    65: invokevirtual #49      // Method java/io/PrintStream.println:(Ljava/lang/String;)V
    68: iconst_1
    69: ireturn
    70: iconst_0
    71: ireturn

public static void main(java.lang.String...);
  Code:
     0: ldc           #55      // String Hello, try to find the username and the [...]
     2: astore_1
     3: aload_1
     4: invokedynamic #57,  0  // InvokeDynamic #1:makeConcatWithConstants:
                               // (Ljava/lang/String;)Ljava/lang/String;
     9: astore_1
    10: getstatic     #41      // Field java/lang/System.out:Ljava/io/PrintStream;
    13: aload_1
    14: iconst_0
    15: anewarray     #2       // class java/lang/Object
    18: invokevirtual #60      // Method java/io/PrintStream.printf:(Ljava/lang/String;
                               // [Ljava/lang/Object;)Ljava/io/PrintStream;
    21: pop
    22: aload_0
    23: arraylength
    24: iconst_2
    25: if_icmpeq     48
    28: getstatic     #41      // Field java/lang/System.out:Ljava/io/PrintStream;
    31: ldc           #64      // String Usage:
    33: invokevirtual #49      // Method java/io/PrintStream.println:(Ljava/lang/String;)V
    36: getstatic     #41      // Field java/lang/System.out:Ljava/io/PrintStream;
    39: ldc           #66      // String java Login username password
    41: invokevirtual #49      // Method java/io/PrintStream.println:(Ljava/lang/String;)V
    44: iconst_m1
    45: invokestatic  #68      // Method java/lang/System.exit:(I)V
    48: aload_0
    49: iconst_0
    50: aaload
    51: aload_0
    52: iconst_1
    53: aaload
    54: invokestatic  #72      // Method access:(Ljava/lang/String;Ljava/lang/String;)Z
    57: ifne          68
```

```
        60: new           #76      // class java/lang/RuntimeException
        63: dup
        64: invokespecial #78      // Method java/lang/RuntimeException."<init>":()V
        67: athrow
        68: getstatic     #41      // Field java/lang/System.out:Ljava/io/PrintStream;
        71: ldc           #79      // String Congratulations, now write the username and [...]
        73: invokevirtual #49      // Method java/io/PrintStream.println:(Ljava/lang/String;)V
        76: getstatic     #41      // Field java/lang/System.out:Ljava/io/PrintStream;
        79: ldc           #81      // String Provide an explanation on how you discover it.
        81: invokevirtual #49      // Method java/io/PrintStream.println:(Ljava/lang/String;)V
        84: goto          113
        87: astore_2
        88: getstatic     #41      // Field java/lang/System.out:Ljava/io/PrintStream;
        91: ldc           #83      // String Wrong username and/or password.
        93: invokevirtual #49      // Method java/io/PrintStream.println:(Ljava/lang/String;)V
        96: getstatic     #41      // Field java/lang/System.out:Ljava/io/PrintStream;
        99: aload_0
       100: iconst_0
       101: aaload
       102: aload_0
       103: iconst_1
       104: aaload
       105: invokedynamic #85,  0  // InvokeDynamic #2:makeConcatWithConstants:
                                   // (Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;
       110: invokevirtual #49      // Method java/io/PrintStream.println:(Ljava/lang/String;)V
       113: return
      Exception table:
         from    to  target type
            48    84    87   Class java/lang/RuntimeException
}
```

We first understand from instructions 28 to 45 of the *main* method that the class is meant to be launched with two parameters, which are namely the username and the password.

We see that from instructions 48 to 54 the *access* static method is invoked with the first command line argument as the first parameter and the second command line argument as the second parameter, which are namely the username and the password provided for login. Then the *access* method checks if its first parameter is equal to the string "admin" at instructions 40 to 46, returning `false` if not. Therefore we can say the the right username is "admin".

We then analyze the *transform* method. Instructions 0 to 4 subtract the constant 97 (the character code of a lowercase "a") to the character given as argument, storing the result in the stack variable position 1. Then, instructions 5 to 7 subtract the constant 1 to the saved variable, instructions 7 to 10 compute the modulo 26 (`% 26`, where 26 is the number of ASCII lowercase letters) of the result and instructions 11 to 13 increment the final result with the constant 97 and returns the result as a character. Instructions 14 to 17 in order store what the method computed in variable 2, loads variable 2 and returns a copy cast to type `char`. Therefore, an equivalent Java source for the method is:

```java
public static char transform(char c) {
    return (char) (((c - ((int) 'a') - 1) % 26) + ((int) 'a'));
}
```

Instructions 0 to 37 show that a string with value "tvvvvqfstuspohqbttxpse" is read character by character the method *transform* is applied, forming a new String. Given the reverse engineered *transform* method and the fact that the string is composed only of lowercase letters starting from "b", both adding and subtracting "a" and computing a modulo 26 of each character are operations that lead to the same character. Indeed, the only change *transform* applies to every single character is to subtract 1, or return the preceding character of each given character. Therefore, given this, we determine the decoded password to be the string "suuuuperstrongpassword".

Exercise E: Monitoring JIT

1. (a) `-XX:-PrintCompilation`
   (b) `-XX:+UnlockDiagnosticVMOptions -XX:+PrintInlining`
   (c) `-XX:NativeMemoryTracking=summary`
2. The output of the command `java -XX:+UnlockDiagnosticVMOptions -XX:+PrintInlining Inlining` is the following:

```
@ 17   java.lang.String::isLatin1 (19 bytes)   inline
@ 27   java.lang.StringLatin1::hashCode (42 bytes)   callee is too large
@ 37   java/lang/StringUTF16::hashCode (not loaded)   not inlineable
@ 6    java.lang.String::coder (15 bytes)   inline
@ 6    Point::<init> (15 bytes)   inline
@ 1    java.lang.Object::<init> (1 bytes)   inline
@ 20   Point::getY (5 bytes)   inline
@ 25   Point::setX (6 bytes)   inline
@ 30   Point::setY (6 bytes)   inline
@ 6    Point::<init> (15 bytes)   inline
@ 1    java.lang.Object::<init> (1 bytes)   inline
@ 20   Point::getY (5 bytes)   inline
@ 25   Point::setX (6 bytes)   inline
@ 30   Point::setY (6 bytes)   inline
@ 20   Point::getY (5 bytes)   accessor
@ 25   Point::setX (6 bytes)   accessor
@ 30   Point::setY (6 bytes)   accessor
```

   Therefore we can confirm that all methods listed were inlined while executing class *Inlining*.

Exercise F: Java Agent

1. The compiled code given cannot be executed as-is as the class *ch.usi.inf.ajp22.Point* contains an invalid java class file format version of 255 (where e.g. for Java 17 the correct value is 61).
2. Please find the relevant source code in folder `ExcerciseF`.