Gradient bitmap calculator analysis

Used metrics:

- Execution time;
- Memory consumption during the execution (heap + stack);

  RSS - resident set size, is the portion of memory occupied by a process that is held in main memory (RAM);
- Code readability.

Language performance

Metric	Kotlin	Go	Dart
Execution time (ms)	7	0.015562	0.088
Max RSS Memory(kb)	36120	1816	13056
Minor (frame) pf	3190	190	1803
Voluntary con. switches	26	3	0
Involuntary con. switches	360	15	7

Kotlin

Considering the elapsed time this one is the slowest one in terms of elapsed time. The same is the for the memory consumption, it uses a greater portion of RAM than both Go and Dart. it may look like the performance is really poor, however it is good to consider the other languages that are not presented in this metric.

Go

The Go language is the fastest among the sample ones. Memory consumption is the best (the lowest) as well.

Dart

The time-memory performance of this programming language lies in between of Kotlin and Go. It is faster and consumes less memory than Kotlin but it is slower and consumes more memory than Go.

Code readability

Remark:

The evaluation of code readability is subjective and may depend from person to person.

Kotlin

```kotlin
import java.util.*
import java.lang.String.*


fun fillGradientLine(arr: kotlin.UIntArray, line: Int, w: Int, a: UByte, b: UByte): Unit {
    val delta = b - a
    val offset = line * w
    for (i in 0 until w) {
        var fraction = i.toFloat() / w.toFloat()
        arr[offset + i] = a.toUByte() + (delta.toFloat() * fraction).toUInt()
    }
}
fun generateGrayGradient(h: Int, w: Int, a: UByte, b: UByte): kotlin.UIntArray{
    val size = w * h
    val arr = kotlin.UIntArray(size)
    for (i in 0 until h){
        fillGradientLine(arr, i, w, a, b)
    }
    return arr
}

fun main(args : Array<String>) {
    val h = 50
    val w = 50
    val a: UByte = 55.toUByte()
    val b: UByte = 233.toUByte()
    val time = System.nanoTime();
    generateGrayGradient(h, w, a, b)
    val timeMilli = (System.nanoTime() - time) / 1000000
    println("$timeMilli")
}
```

Kotlin maintain the java naming for classes and uses java libraries. It is easy to read for those, who get used to C and Java programming languages. The end line separators (semi columns) are not obligatory in Kotlin, so there will be less mistakes that are connected to end line typos. However, due to this "not obligatory semi columns" thing there is a huge probability of messy code with mixed style line. One may want to add end line separators, the others do not.

The external functionality: kotlin uses java libraries for input/output operations and for some types (Strings, for example). In addition, now (april 2020), unsigned byte types as well as arrays are running under "experimental mode".

The length of the Kotlin code is approximately the same as Go code (31 lines Kotlin vs 32 lines Go) among the represented ones (31 lines). It is worth to mention that it became shorter than Java code and it is easier to read. Comparing to Go and Dart Kotlin code seems to be cleaner and more human-readable.

Go

```go
package main
import (
    "fmt"
    "time")

```

```go
func fillGradientLine(arr []uint8, line int, w int, a uint8, b uint8) {
    delta := b - a
    offset := line * w
    for i := 0; i < w; i++ {
        fraction := float32(i) / float32(w)
        arr[offset + i] = a + uint8(float32(delta) * fraction)
    }
}
func generateGrayGradient(h int, w int, a uint8, b uint8) []uint8 {
    size := w * h
    arr := make([]uint8, size)
    for i := 0 ; i < h; i++ {
        fillGradientLine(arr, i, w, a, b)
    }
    return arr
}

func main() {
    h := 50
    w := 50
    a := uint8(55)
    b := uint8(233)
    start := time.Now()
    generateGrayGradient(h, w, a, b)
    elapsed := float32(time.Since(start).Nanoseconds()) / 1000000
    fmt.Printf("%f\n", elapsed)
}
```

The length of the code is approximately the same as the Kotlin code. The good thing about go is that there are no any end line separators and at the same time functions are wrapped into curly braces, so the code is easier to read. It is clear where the function starts and ends. On the other hand, the variable declaration with ':=' sign seems to be too complicated. In addition, speaking about declarations: to declare an array one must put brackets before the type of the array content. It is interesting decision but from my point of view this makes the code more difficult to read. Speaking about external functionality: go uses "fmt" package for formatted input/output operations (analogous to C's printf and scanf).

In general code is "not great, not terrible". It is not understandable for human and number of special symbols is low.

Dart

```dart
void fillGradientLine(arr, line, w, a, b){
    var delta = b - a;
    var offset = line * w;
    for(var i = 0; i < w; i++){
        var fraction = i / w;
        arr[offset + i] = a + (delta * fraction).floor();
    }
}

List generateGrayGradient(h, w, a, b){
    var size = h * w;
    var arr = [];
    for(var i = 0; i < size; i++) arr.add(0);
```

```
    for(var i = 0; i < h; i++) fillGradientLine(arr, i, w, a, b);
    return arr;
}

void main() {
    var h = 10;
    var w = 10;
    var a = 55;
    var b = 233;
    Stopwatch stopwatch = new Stopwatch()..start();
    generateGrayGradient(h, w, a, b);
    print(stopwatch.elapsedMicroseconds /1000); // executing time in
milliseconds
}
```

Dart code is the shortest among the sample ones (26 lines). Although there are end line separators (semi columns), the code seems to be clean. It is easy to read. The functions are wrapped in the curly braces, as in many other programming languages. The names of built-in types are short and clearly state what the type is. The external functionality: dart does not require additional imports to perform input/output operations. print function is built-in.