TÉCNICO LISBOA

Autoboxing in Java

Professor:

António Menezes Leitão

André Rodrigues nº69998
Gonçalo Castilho nº75305
Sílvia Timóteo nº75770

TÉCNICO LISBOA

Introduction

- Since Java 1.5 there exists boxing and unboxing operations. These consist in converting primitive types into their corresponding wrapper(reference) type and the other way around. Although this is done automaticaly it is very costly.

TÉCNICO LISBOA

Goals

- Detecting boxing and unboxing operations in Java for all primitive types and their respective reference types.

Solution

TÉCNICO LISBOA

Code Instrumentation

- First we added a TreeMap to count the number of boxing/unboxing operations done ordered by the following keys:

  Method Name->Type->Action->Count

Action={Boxing,Unboxing}

TÉCNICO LISBOA

Code Instrumentation

- Then we injected a function(*insertData*), to increment the count in the TreeMap;

- Also we injected another function(*printData*) to print the results.

TÉCNICO LISBOA

Algorithm used

1. Search for where boxing and unboxing operations where done in the code;
2. Added a call after the operation to function insertData;
3. Added a call after the main method to the function printData.

TÉCNICO LISBOA

Dificulties

- Parametrization(Javassist);
- Stack overflow;

TÉCNICO LISBOA

Extensions added

TÉCNICO LISBOA

Motivation

- Autoboxing operations are relatively slower than simply calling a new <Object>() method;
- Autoboxing caches reference objects and uses these cached objects instead of always instantiating a new Object;
- Calling new <Object>() is a little bit faster than autoboxing (although it uses more memory.

TÉCNICO LISBOA

Solution

- To improve execution times in programs we replace Autoboxing operations for the call to the respective new method of each reference type.

Results

TÉCNICO LISBOA

Old output format

```
C:\Users\gonca\Downloads\g13>java -cp BoxingProfiler.jar ist.meic.pa.BoxingProfiler SumIntegers
Sum: 124750
Time: 7
SumIntegers.main(java.lang.String[]) boxed 2 java.lang.Integer
SumIntegers.main(java.lang.String[]) boxed 1 java.lang.Long
SumIntegers.printSum(java.lang.Long) unboxed 1 java.lang.Long
SumIntegers.sumOfIntegerUptoN(java.lang.Integer) unboxed 501 java.lang.Integer
SumIntegers.sumOfIntegerUptoN(java.lang.Integer) boxed 501 java.lang.Long
SumIntegers.sumOfIntegerUptoN(java.lang.Integer) unboxed 501 java.lang.Long
```

TÉCNICO LISBOA

New output format

```
C:\Users\gonca\Downloads\g13>java -cp BoxingProfiler.jar ist.meic.pa.BoxingProfilerExtended SumIntegers
Sum: 124750
Time: 5
SumIntegers.printSum(java.lang.Long) unboxed 1 java.lang.Long
SumIntegers.sumOfIntegerUptoN(java.lang.Integer) unboxed 501 java.lang.Integer
SumIntegers.sumOfIntegerUptoN(java.lang.Integer) unboxed 501 java.lang.Long
Replaced boxing operation in SumIntegers.main(java.lang.String[]) 2 times for type: java.lang.Integer
Replaced boxing operation in SumIntegers.main(java.lang.String[]) 1 times for type: java.lang.Long
Replaced boxing operation in SumIntegers.sumOfIntegerUptoN(java.lang.Integer) 2 times for type: java.lang.Long
```

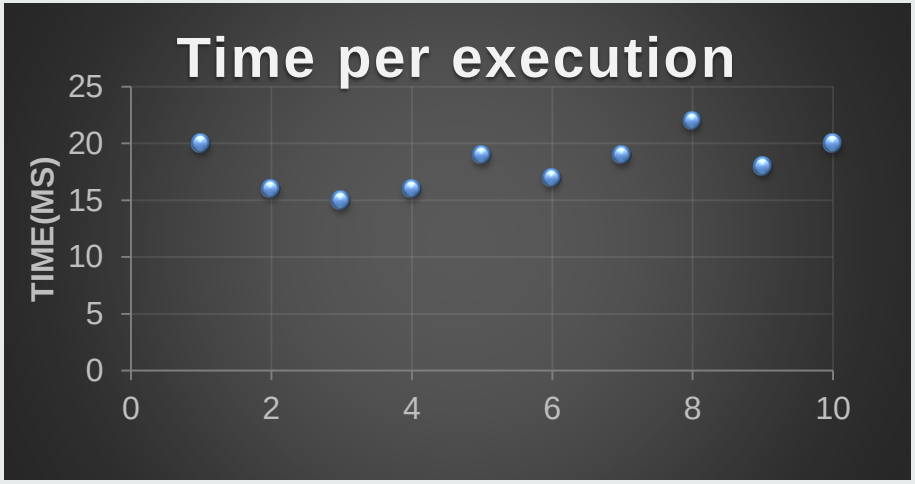
TÉCNICO LISBOA

Test Program

```
public class SumIntegers {
    private static long sumOfIntegerUptoN(Integer n) {
        Long sum = 0L;
        for (int i = 0; i < n; i++) {
            sum += i;
        }
        return sum;
    }
    private static long printSum(Long n) {
        System.out.println("Sum: " + n);
        return n;
    }
    public static void main(String[] args) {
        long start = System.currentTimeMillis();
        printSum(sumOfIntegerUptoN(10000));
        Integer i=1;
        long end = System.currentTimeMillis();
        System.out.println("Time: " + (end - start));
    }
}
```

BoxingProfiler(original)

- The average execution time of the program is 18.2ms with 22ms being the highest execution time and 15ms being the smallest.



BoxingProfilerExtended

- The average execution time of the program is 14ms with 16ms being the highest execution time and 12ms being the smallest.