Programmeerimise põhikursus Javas

Loeng 15

http://courses.cs.ttu.ee/pages/ITI0011

Outline

- Homework stuff

- Test results
- Exams
- codingbat?

- IV homework - Gomoku

- Timing for Gomoku
- Minimax revisited
- Recursion

Homework submission

- https://courses.cs.ttu.ee/pages/ITI0011:git

- Homeworks into HW1, HW2, HW3 and HW4 folders
- **Check your score table to see git status**
- Homework 4 to be pushed into git latest **December 14th 23:59**
  - into folder "HW4"
- **The last option to defend HW4 is in December**

- **Course code example in git:** http://firstname.lastname@git.ttu.ee/kursused/iti0011/materjalid.git
- Use UNI-ID to access materials (not visible in browser)

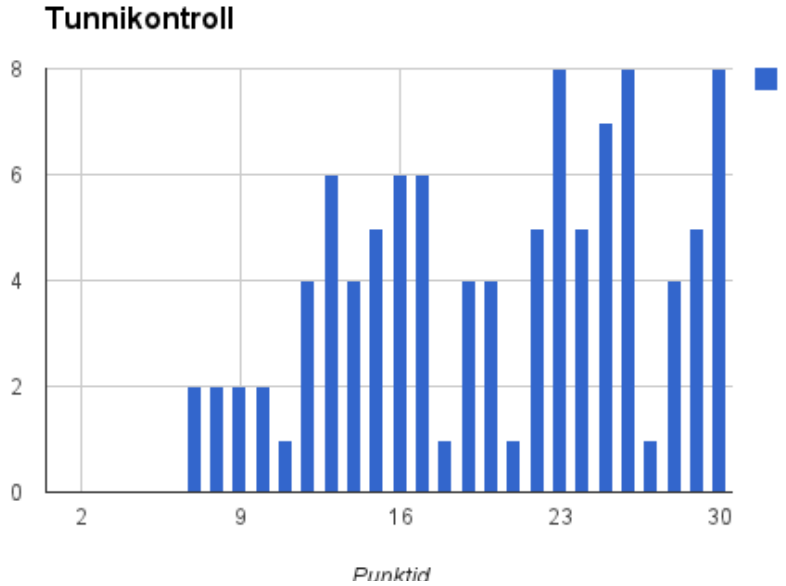
Exam

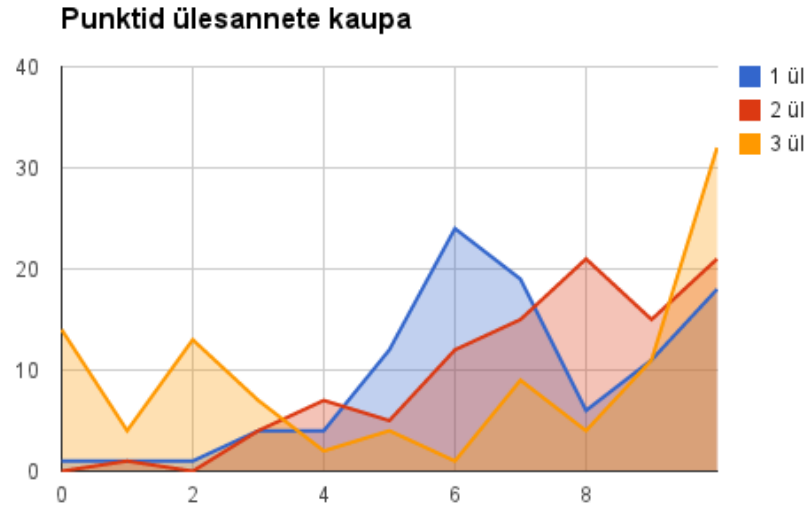
- Exam times:
  - **08.01.2015 (Thursday) 10:00 and 13:00 U06A-201**
  - **16.01.2015 (Friday) 10:00 and 13:00 U06A-229**
- At least 3 homeworks should be defended + pushed to git
- Exam:
  - written **on paper**
  - **2h time**
  - The time (10 or 13) is assigned to every student (check points table)
  - **Identification document** (ID-card, passport, drivers licence)
- Consultations:
  - **07.01 (Wednesday) 10:00 ICT-411**
  - **15.01 (Thursday) 10:00 ICT-411**
  - Register in doodle

Test results

- **101 students**
- 3 "questions", each max 10 points
- **8 students got 30p**
- 33 students got 25p or more
- 56 students got 20p or more
- 28 students got 15p or less

- 1 question avg 6.88
- 2 question avg 7.49
- **3 question avg 5.93**
- Total avg 20.3





Task 1 - reading code

- function calls! A lot of students "forgot" about the second "foo":

```
public static void main() {
	foo();
	System.out.println("main");
	bar(foo());
}

public static void foo() {
	System.out.println("foo");
}

public static void bar() {
	System.out.println("bar");
}
```

Output:

foo
main
foo
bar

Scoping

- **A variable is accessible everywhere within the set of curly braces where it is declared, including code within nested curly braces**

Variable scope - block

- Variable exists from the **declaration** until the **end of the block**:

```
{
    // no c variable here
    int c = 20;
    System.out.println(c);
} // c is freed here
```

Variable scope - block

- Variables is accessible in nested blocks:

```
{
        // no c variable here
        int c = 20;
        System.out.println(c);
        {
                // no variable d here
                int d = 100;
                System.out.println(c);
                System.out.println(d);
        } // d is freed here, c still exists
} // c is freed here
```

Variable scope - block

- Not allowed to have multiple local variables with the same name within the same scope:

```
{
	// no c variable here
	int c = 20;
	System.out.println(c);
	{
		// c already exists
		int c = 100; // not allowed!
		System.out.println(c);
	} //
} // c is freed here
```

Variable scope - loops

- Loop is a block
- Variable declared inside loop header is accessible only inside the loop (until the end of the block):

```
for (int i = 0; i < 10; i += 2) {
      // i is defined in the header,
      // and is accessible inside the block
      System.out.println(i);
} // i is freed here
```

Variable scope - method declaration

- Method is a block
- Method parameters are declared variables:

```
public static int foo(int a, int b) {
        // a and b are declared as int variables
        // and values are passed when the method is called
        return a + b;
} // a and b are freed here
```

Variable scope - class/instance variables

- Class / instance variables are accessible throughout the whole class (block), even before the declaration:

```
public class NumberExample {
	public void printNumber() {
		// although number is not declared yet,
		// it is declared within the class definition (below)
		System.out.println("number is: " + number);
	}
	// number is accessible everywhere in the class,
	// even on the lines before the declaration line
	public int number = 5;
}
```

Variable scope

- The scope of variable starts with its most recent declaration
- Local variables take precedence over class variables:

```
public class LocalVsClassVariables {
        public int number = 10; // instance variable
        public void printNumber() {
                // instance variable "number" is used,
                System.out.println(number); // outputs 10
        }

        public void printMyNumber(int number) {
                // variable number refers to the argument
                // passed to the method
                System.out.println(number);
                // to access instance variable, use "this"
                System.out.println(this.number);
        }
        public void printLocalNumber() {
                int number = 5; // local variable defined here
                System.out.println(number); // outputs 5
        }
}
```

Exercise

- What is the output?

```
public class VariableScopeExercise {
            public static int a = 10;
            public static int b = 2;
            public static void main(String[] args) {
                        System.out.println("1 a:" + a);
                        System.out.println(foo(a));

                        System.out.println("2 a:" + a);
                        System.out.println(foo(a));

                        System.out.println(foo(bar(b)));
                        System.out.println("3 a:" + a);
                        System.out.println("b: " + b);

            }
            public static int foo(int a) {
                        a++;
                        b++;
                        return a;
            }
            public static int bar(int b) {
                        a++;
                        b++;
                        return a;
            }
}
```

Increment/decrement operators

- Postfix form:
  `x++` Increments value of x by 1 (x = x + 1)
  `x--` Decrements value of x by 1 (x = x - 1)

- Can also be written in (prefix) form:
  `++x` and
  `--x`

- The difference is:
  - `result++` evaluates to the **original** value
  - `++result` evaluates to the **incremented** value

Increment/decrement operators

```
int a = 1;


System.out.println(a); // outputs 1
a++; // a = a + 1
System.out.println(a); // outputs 2
System.out.println(a++); // outputs 2, a = a + 1
System.out.println(a); // outputs 3
System.out.println(++a); // a = a + 1, outputs 4
```

- In case of `a++`, the original value (2) is returned and a is **incremented after**
- In case of `++a`, the a is **incremented before** and the new value (3) is returned

Assignment operators

- `x += 5` same as `x = x + 5`
- `x -= 5` same as `x = x - 5`
- `x *= 5` same as `x = x * 5`
- `x /= 5` same as `x = x / 5`

```
a = 15;
System.out.println(a *= 2); // a = a * 2, outputs 30
System.out.println(a /= 3); // a = a / 3, outputs 10
System.out.println(a); // outputs 10
```

Method overloading

- Java allows methods with the same name as long as the arguments are different
  - The number of arguments can be different
  - The type of arguments can be different

```
public static int foo() {
            // when called foo();
            return 0;
}
public static int foo(int a) {
            // when called foo(1);
            return a;
}
public static int foo(int a, int b) {
            // when called foo(1, 2);
            return a + b;
}
public static double foo(double a) {
            // when called foo(1.5);
            return a / 2;
}
```

Method calls

- Before calling the method, arguments have to be evaluated
- For the line:

```
foo(bar(a + 3))
```

- The following takes place (in the given order):
  - `a + 3` is calculated
  - the result of `a + 3` is passed to `bar` method
  - the result of `bar(a + 3)` is passed to `foo` method

Exercise

- What is the output?

```
public class Saladus {
   public static int x=2;
   public static int y=2;

   public static void main(String[] args) {
     foo(bar(foo(x)));
     System.out.println("main x,y: "+x+" "+y);
   }

   public static int foo(int x) {
     x++;
     y++;
     System.out.println("foo x,y: "+x+" "+y);
     return x;
   }

   public static int bar(int x) {
     int z=0, y=10, u=0;
     --y;
     for(y=1; y<(x*x); y++) {
       for(z=1; z<x; z++) {
            u++;
       }
     }
     System.out.println("bar x,y: "+x+" "+y);
     return z;
   }
}
```

Task 1 - let's try again

```
public class ReadMe {
  public static int a = 0;
  public static int b = 0;

  public static void main(String[] args) {
    a++;
    foo();
    System.out.println("main a:" + a + " b:" + b);
    bar(foo());
  }

  public static int foo() {
    int a = 1;
    a++;
    b++;
    System.out.println("foo a:" + (++a) + " b:" + b);
    return a;
  }

  public static void bar(int a) {
    a++;
    for (int i = 0; i < 10; i++) {
      a += i % 3;
    }
    System.out.println("bar a:" + a + " b:" + b);
    System.out.println("traktor".substring(4, 7));
  }
}
```

Output:

foo a:3 b:1

main a:1 b:1

foo a:3 b:2

bar a:13 b:2

tor

Task 2.1

```
ArrayList<Integer> numbers = new
ArrayList<Integer>();
numbers.add(1);
numbers.add(1);
numbers.add(2);
```

numbers.size()? 3

Task 2.2

```
ArrayList<Integer> numbers = new
ArrayList<Integer>();
numbers.add(1);
numbers.add(1);
numbers.add(2);

numbers.remove(1);

numbers.size()? 2
```

Task 2.3

int[] numbers = new int[3];
number.length? 3

```
int[] numbers = new int[3];
numbers[0] = 1;
numbers[1] = 3;
```

numbers.length? 3

Task 2.4

```
String a = "Teretulemast!";
```

a.substring(3,5)? "et"

a.indexOf("e")? 1

a.charAt(8)? 'm'

Task 2.5

```
"1" + 1? "11"
```

Task 2.6

```
int sum = 0;
for (int i = 0; i < 5; i++) {
      if (i > 3) break;
      sum += i;
}
```

sum? 6

Task 2.7

```
int sum = 0;
for (int i = 0; i < 8; i++) {
    if (i % 2 == 1) continue;
    sum += i;
}
```

sum? 12

Task 2.8

```
int i = 0;
while (i > 2) {
    i++;
}
```

i? 0

Task 2.9

```
int i = 0;
do {
    i++;
} while (i > 2);
```

i? 1

Task 2.10

```
int[][] matrix =
          {{0, 1, 0}, {1, 1, 0}, {-1, -1, -1}};
```

matrix[1][2]? 0

matrix.length? 3

Task 3

- check the last word (in case of splitting by spaces)
- you have method declaration:
```
public static int longestWordStart(String str) {
```
- No need to ask user input or define your own method
- method also has a return type int
- No need to print out the result, instead return it

Remarks

- if (str.substring(x, y).length() > longest)
  => (y - x) > longest
- int a = str.indexOf(str.charAt(i));
- Variable names (do not use o, i, l etc): if (str.length > o)
- Finding MAX (not correct):

```
int max = 0;
for (int i = 0; i < massiv.length; i++) {
  if (massiv[i] > massiiv[i + 1]) max = massiiv[i];
  else max = massiiv[i + 1];
}
```

Measure time (for Gomoku)

- Use System.nanoTime():

```
long startTime = System.nanoTime();
// ... the code being measured ...
long estimatedTime = System.nanoTime() - startTime;
```

- When getMove() is called, mark the current time as startTime
- during minimax, check the lapsed time
- If the lapsed time is high, start ending your algorithm:

```
if (System.nanoTime() - startTime > ALLOWED_MOVE_TIME)
return getScore(...)
```

- Note, that even if you return, it still takes additional time to return from all the branches