Lecture 04: Loops

SE115: Introduction to Programming I

Previously on SE 115…

- Flow control
- if
- if else
- if, else if, else
- case, break, default
- Did we miss something?

Ternary Operator

- Shorthand if… else
- “Ternary” because three operands are needed:
  - a “condition”, a result for “true” and a result for “false”.
- Replace multiple lines of code with a single line
- Syntax:
  - `variable = (condition) ? expressionTrue : expressionFalse;`

Ternary Operator

Example:

```
int temperature = 32;
if (temperature < 25) {
  System.out.println("Colder than room temperature.");
} else {
  System.out.println("Warmer than room temperature.");
}
```

With Ternary operator:

```
int temperature = 32;
String result = (temperature < 25) ? "Colder than room temperature." : "Warmer than room temperature.";

System.out.println(result);
```

Loops

- A program flows top to bottom.
- However, last week we have learned that we can create branches in the flow.
- Today we will learn how to loop certain parts of the code.
- Loops are blocks of code that are executed repeatedly.
- Here’s a problem:
    - We want to find the average grade in a class. How can we write this program?

Reasoning

- To write any program, we need to know the solution, so that we can write it in the programming language.
- Calculating the average is simple: (1) get the sum of the values, (2) then divide the sum by the number of values.
- To realize this program, we have to write it step by step using the programming constructs that we know.

Reasoning

- We can ask for the grade of each student. This can be stored in a temporary variable.
- We have to keep track of the “number of students” and the “sum”.
- Here is an attempt:
  - Create int variable “sum” and initialize it to 0.
  - Create int variable “num” and initialize it to 0. This is the number of students.
  - Create int variable “grade” and initialize it to 0. This is the grade for each student.
  - Input the grade of the first student, store it in **grade**, increment **sum** with grade, increment **num** with 1.
  - Input the grade of the second student, store it in **grade**, increment **sum** with grade, increment **num** with 2.
  - Input the grade of the third student, store it in **grade**, increment **sum** with grade, increment **num** with 3.
  - Calculate **sum** / **num** and print it as the average.
- Let me implement it.

```java
import java.util.Scanner;

public class GradeAverage {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int sum = 0;
        int num = 0;
        int grade = 0;

        // first
        System.out.print("Enter the grade for the student: " );
        grade = sc.nextInt();
        sum += grade;
        num++;

        // second
        System.out.print("Enter the grade for the student: " );
        grade = sc.nextInt();
        sum += grade;
        num++;

        // third
        System.out.print("Enter the grade for the student: " );
        grade = sc.nextInt();
        sum += grade;
        num++;

        System.out.println((float)sum / (float)num);
    }
}
```

Problems

- Of course, I have copied and pasted the part which gets the grades.
- If the number of students change, so does the code.
  - Compile and run again.
- How can loops help?
- First, identify the repetition, then use the loop construct.

```
int num = 0;
int grade = 0;

// first
System.out.print("Enter the grade for the student: ");
grade = sc.nextInt();
sum += grade;
num++;

// second
System.out.print("Enter the grade for the student: ");
grade = sc.nextInt();
sum += grade;
num++;

// third
System.out.print("Enter the grade for the student: ");
grade = sc.nextInt();
sum += grade;
num++;
```

Loops

- A loop is made up of three parts:
  - The initial value(s),
  - The step,
  - The stopping condition
- Any kind of loop requires all of them in some way or another.
- The stopping condition is important: if it is never met, the loop does not stop. We call such loops as **infinite loops.**
- Let’s begin with the “for” loop.

“for” loop

- The **for** loop has the following structure:

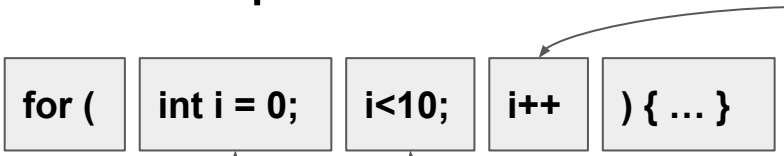
```
for([initial value] ; [stopping condition] ; [step]) {
  // body of the loop
}
```

- Notice that the **for** loop contains all three conditions in its definition.
- Also notice that the “body of the loop” is in a block {}.
- Here is an example:

```
for(int i=0;i<10;i++) { … }
```

- Let me break it down for you.

“for” loop



This is the **initial value**. Notice that it is defined here, not outside the loop. So the variable **i** will be available in the loop body.

This is the **stopping condition**. The loop will continue to run (to loop) as long as this is true. When i=0, at the very beginning, 0 is less than 10, so the loop will run.

This is the **step**; once the loop body is executed, and we reach the closing brace }, we increment i by 1. After 10 loops, the stopping condition, i<10, will be false, and the loop will exit.

“for” loop

- Let’s use **for** loop for something.
- Let’s print SE115 100 times on the screen!

```
public class Hundred {
  public static void main(String[] args) {
    for(int i=0;i<100;i++) {
      System.out.println(“SE115”);
    }
  }
}
```

“for” loop

- Let’s use it for something useful.
- How about the average grade example?
- We can ask the user “How many students are there?”
- Then, we can loop that many times to get the student grades.
- Let’s first think about it; here’s the previous one:

“for” loop

- Create int variable “sum” and initialize it to 0.
- Create int variable “num” and initialize it to 0. This is the number of students.
- Create int variable “grade” and initialize it to 0. This is the grade for each student.
- Input the grade of the first student, store it in **grade**, increment **sum** with grade, increment **num** with 1.
- Input the grade of the second student, store it in **grade**, increment **sum** with grade, increment **num** with 2.
- Input the grade of the third student, store it in **grade**, increment **sum** with grade, increment **num** with 3.
- Calculate **sum** / **num** and print it as the average.

“for” loop

- Create int variable “sum” and initialize it to 0.
- Create int variable “num” and initialize it to 0. This is the number of students.
- Create int variable “grade” and initialize it to 0. This is the grade for each student.
- Ask the user the number of students, and store it in variable **num**.
- Create a **for** loop that will run **num** number of times.
    - Input the grade of the corresponding student, store it in **grade**, increment **sum** with grade.
- Calculate **sum** / **num** and print it as the average.

“for” loop

```
import java.util.Scanner;

public class GradeFor {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int sum = 0;
        int num = 0;
        int grade = 0;

        System.out.print("Enter the number of students: ");
        num = sc.nextInt();

        for(int i=0;i<num;i++) {
            System.out.print("Enter the grade for student " + (i+1) + ": ");
            grade = sc.nextInt();
            sum += grade;
        }

        System.out.println((float)sum / (float)num);
    }
}
```

See what I did here?
It will print
Enter the grade for student 1:
for student i=0…

Remarks

- Yes, we always start from 0. Computer science demands it.
- The variable i is only available in the block { } - this is called the “scope.”
  - In simple terms, the variable lives in the block it is created. This is true for all the {} blocks we use; class definitions, main function, if clauses, and for loops.
  - If you define something in a block, it lives in that block, and cannot survive once it sees the closing brace } of that block.
- So, if you define a variable in the loop body, it will not survive outside of it.

Looping tricks

- You can omit any part of the for loop definition;

```
for(int i=0; ;i++) { … } // no stopping condition!

// only the stopping condition! assuming i has been declared.
for(; i<10; ) { … }

for(; ;) { … } // nothing! infinite loop!
```

- So, let me show you a few cases.

Looping tricks

- Getting out of the loop before the stopping condition is met:
  - This should be obvious, since the loop can begin without a stopping condition.
  - In that case, we can use “break” to get out of the loop.

```
for(int val=0; ;) { // no stopping cond, no increment
  val = sc.nextInt();
  if (val == 100) {
    break;
  }
}
```

Get a value for variable val, if the user enters 100, exit the loop.

You can use an additional variable to find out how many time we have looped. How can it be done?

Looping tricks

- In some buildings, 13th floor is not used, because the number 13 is believed to be unlucky. So the elevator skips that floor.
- Can we skip some of the loop body? Yes, by using “continue”.
- When the compiler sees the “continue” keyword, it skips the rest of the loop body, and it continues to the loop with the next step.
- Here is one example.

Looping tricks

```
public class ContinueLoop {
  public static void main(String[] args) {
    for(int i=0;i<20;i++) {
      System.out.println("Working on number: " + i);
      if(i==13) {
        continue;
      }
      System.out.println("Processing number " + i);
      System.out.println("Done!");
    }
  }
}
```

See how it skips 13?

```
Working on number: 12
Processing number 12
Done!
Working on number: 13
Working on number: 14
Processing number 14
Done!
Working on number: 15
Processing number 15
```

Looping tricks

- Both **break** and **continue** work on other loop structures, too.
- Speaking of other loops structures, let's learn the next one: the **while** loop.
- I'm sure some of you asked me "what if we don't know the number of loops?"
- That is a very important question, and the **while** loop can help.

“while” loop

- The “while” loop has a much more simple structure.

```
while (condition) {
  // loop body
}
```

- As long as the “condition” is true, the loop will continue to run.

```
while(true) { ... } // an infinite loop, can we get out?
```

- So, what about the initial value and the step?
- Well, they are defined outside of the definition of the while loop.
- The step can be forgotten; it is usually the last line in the while loop body.

“while” loop

- Let’s make use of the while loop to improve our average grade program.
- Instead of asking for the number of students, let’s stop when the user enters a negative value.
- So the loop will continue until we reach the negative value, then we will “break” and then calculate the average.
- This creates a few challenges; the negative value must not be added to the **sum** variable, and the number of students should not be incremented.
    - We can use “if” clauses to handle them.

“while” loop

- Create int variable “sum” and initialize it to 0.
- Create int variable “num” and initialize it to 0. This is the number of students.
- Create int variable “grade” and initialize it to 0. This is the grade for each student.
- Tell the user to enter a negative value to stop entering values.
- Create a **while** loop that will run while grade >=0.
    - Input the grade of the corresponding student, store it in **grade**.
    - If **grade** is negative, then skip the rest of the loop with **continue.**
    - Increment **sum** with grade, increment **num** with 1.
- Calculate **sum** / **num** and print it as the average.

“while” loop

```
import java.util.Scanner;

public class GradeWhile {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int sum = 0;
        int num = 0;
        int grade = 0;

        System.out.println("Enter a negative grade to stop entering grades.");

        while(grade >= 0) {
            System.out.print("Enter the grade for student "+ (num+1) + ": ");
            grade = sc.nextInt();
            if (grade < 0) {
                continue;
            }
            sum += grade;
            num++;
        }

        System.out.println((float)sum / (float)num);
    }
}
```

initial value

condition

continue makes the code skip the two lines below

step

“while” loop

- Create int variable “sum” and initialize it to 0.
- Create int variable “num” and initialize it to 0. This is the number of students.
- Create int variable “grade” and initialize it to 0. This is the grade for each student.
- Tell the user to enter a negative value to stop entering values.
- Create a **while** loop that will run **forever**.
  - Input the grade of the corresponding student, store it in **grade**.
  - If **grade** is negative, then exit the loop with **break.**
  - Increment **sum** with grade, increment **num** with 1.
- Calculate **sum** / **num** and print it as the average.

“while” loop

```
import java.util.Scanner;

public class GradeWhileBreak {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int sum = 0;
        int num = 0;
        int grade = 0;

        System.out.println("Enter a negative grade to stop entering grades.");

        while(true) {
            System.out.print("Enter the grade for student "+ (num+1) + ": ");
            grade = sc.nextInt();
            if (grade < 0) {
                break;
            }
            sum += grade;
            num++;
        }

        System.out.println((float)sum / (float)num);
    }
}
```

infinite loop

stopping infinite loop

“while” loop

- Which one should I use, “for” loop or “while” loop?
- It depends on the problem.
- If we don’t know the number of loops, such as in this case where the user might come up with different number of students, or when we are reading from a file or network source, then “while” provides a mechanism to stop when a specific condition is given.
- The “for” loop is handy when we know the number of loops, and when we need the loop variable (the variable i) to perform an operation.
    - For example, when we switched to “while” loop in the average grade problem, we no longer needed the i variable.

“do…while” loop

- There is one more loop structure:

```
do {
  // loop body
} while([condition]);
```

Notice the semicolon here!

- When you look at it I believe you can understand how this works.
- We run the loop body while the condition is true.
- The difference between “do-while” and “while” is that “do-while” evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within the “do” block **are always executed at least once**.
- This is a good opportunity to rewrite the grade average problem.

“do…while” loop

```
import java.util.Scanner;

public class GradeDoWhile {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int sum = 0;
        int num = 0;
        int grade = 0;

        System.out.println("Enter a negative grade to stop entering grades.");

        do {
            System.out.print("Enter the grade for student "+ (num+1) + ": ");
            grade = sc.nextInt();
            if (grade >= 0) {
                sum += grade;
                num++;
            }
        } while(grade >= 0);

        System.out.println((float)sum / (float)num);
    }
}
```

Pay attention to this part. I did not use **break** or **continue**, but I have to make sure that a negative value is not counted among the student grades.

Any problems?

- All versions of the average grading has one problem which is very hard to see at first.
- Think about it, and I'll let you know at the end of the lecture.

Nested Loops

- We have used an if clause within an if clause or a loop.
- Can we use a loop within a loop? What happens then?
- We call such structures “nested loops”.

```
for(int i=0;i<10;i++) {
  for(int j=0;j<10;j++) {
    // do something
  }
}
```

For each loop of “i,” the “j” for loop runs 10 times. Since “i” also runs 10 times, the body for “j” loop runs 100 times.

Nested Loops

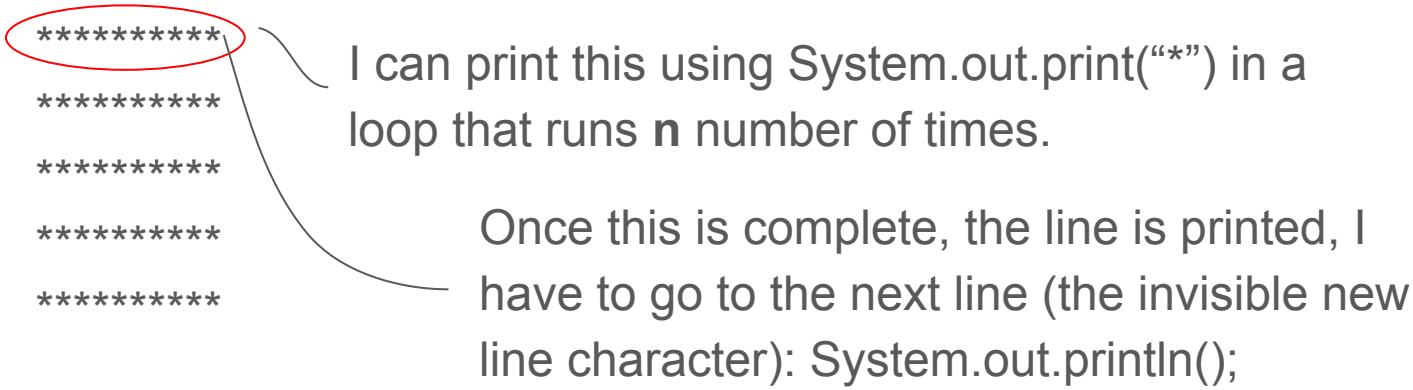
Here’s a visual example. I want to print a square of stars.

****
****
****
****
****

How do I do that? Each line has 10 stars, there are 5 lines, but assume that these values are entered by the user, so the user can enter 6 stars and 19 lines, or anything they like.

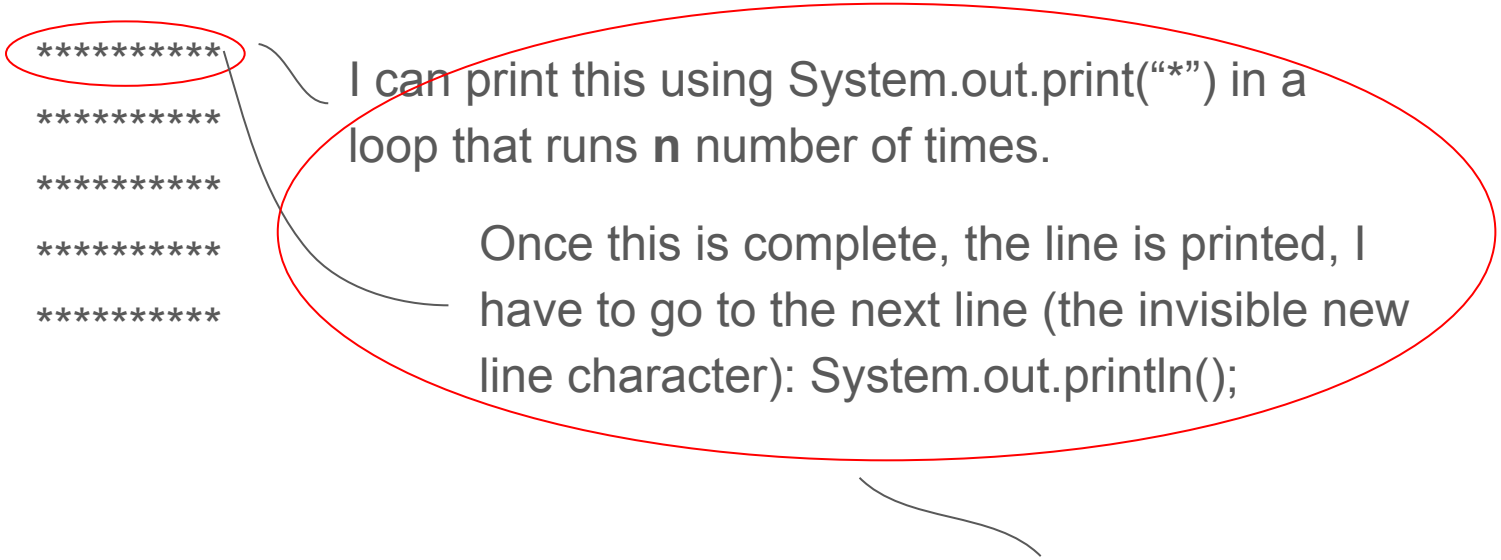
Nested Loops

Each line will have **n** number of stars, and there will be **m** number of lines.



Nested Loops

Each line will have **n** number of stars, and there will be **m** number of lines.



Nested Loops

```java
import java.util.Scanner;

public class NestedLoops {
    public static void main(String[] args) {
        Scanner sc  = new Scanner(System.in);
        int n = 0;
        int m = 0;

        System.out.print("Enter the number of stars: ");
        n = sc.nextInt();
        System.out.print("Enter the number of lines: ");
        m = sc.nextInt();

        for(int i=0;i<m;i++) {
            // this is the outer loop, the lines
            for(int j=0;j<n;j++) {
                // print stars here
                System.out.print("*");
            }
            // Once the stars are printed, print \n
            // for the next line.
            System.out.println();
        } // this should do it.
    }
}
```

```
❯ javac NestedLoops.java
❯ java NestedLoops
Enter the number of stars: 19
Enter the number of lines: 8
*******************
*******************
*******************
*******************
*******************
*******************
*******************
*******************
```

Game Programming

- OK, let’s have more fun than printing lines.
- Since you know “if” clauses and loops, let’s implement a “number guessing games”.
- Java can generate random numbers using the Random class.
- We will let the computer generate a random value between a range, and then we will try to guess it.
- For each attempt, the computer will either say “go higher,” “go lower,” or “you guessed it!”
- Then, it will print the number of times we were able to find the value.
- Let’s think about it together.

Game Programming

- Let the computer generate a random value, r.
  - Don’t worry how it does it, I’ll show the code.
- Set the number of attempts to 0.
- Do the following…
  - Ask the user for a number,
  - Increment number of attempts,
  - If it is greater than r, print “go lower”,
  - If it is lower than r, print “go higher”,
  - If it is equal to r, print “you guessed it!”
- …while the number the user enters is not equal to r.

Game Programming

```java
import java.util.Scanner;
import java.util.Random;

public class LoopGuess {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Random rd = new Random(System.currentTimeMillis());
        int r = rd.nextInt(100); // up to 100
        int n = -1; // user value
        int attempts = 0;

        do {
            System.out.print("Guess my number: ");
            n = sc.nextInt();
            attempts++;
            if(n > r) {
                System.out.println("Go lower!");
            } else if (n < r) {
                System.out.println("Go higher!");
            } else {
                System.out.println("You found it!");
            }
        } while(r != n);
        System.out.println("You did it in " + attempts + " attempts!");
    }
}
```

```
❯ javac LoopGuess.java
❯ java LoopGuess
Guess my number: 50
Go higher!
Guess my number: 75
Go higher!
Guess my number: 82
Go higher!
Guess my number: 90
Go higher!
Guess my number: 95
You found it!
You did it in 5 attempts!
```

Homework

- How about some artificial intelligence?
- You hold a number in your head.
- The program will ask you for an upper limit.
- Then, it will start guessing your number, you respond by
  - 0 -> the number is correct,
  - 1 -> the number is greater than your number,
  - -1 -> the number is smaller than your number.
  - Do not lie to your computer!
- The program must be clever enough to keep the number of attempts as few as possible!

Homework

- Find the maximum and the minimum:
  - Let the user enter any number of positive integer values, until a negative one is entered.
  - Among these numbers, print the maximum and minimum values.
  - Very challenging!
- Consecutive numbers:
  - Let the user enter any number of positive integer values, until a negative one is entered.
  - Tell the user if the last two numbers are consecutive, as in 41 and 42.
- Greater or smaller:
  - Again, let the user enter any number of positive integer values, until a negative one is entered.
  - Tell the user the number they have entered is greater or smaller than the previous one.