Lecture 02: Values and Variables

SE115: Introduction to Programming I

Previously on SE115…

- Introduction
- Hardware / Software
- Text and binary files
- Compilers
- Programming and Programming Languages
- Hello world!

“Hello, World” Revisited

Let’s have a recap. Here’s file **Hello.java** once again.

The file name and the public class name must be the same.

main: Entry point to the program. It is static because we don’t need an object of the Hello class.

```java
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello, world");
  }
}
```

println: Method for printing on the standard output.

The parameter for println. It is a String, a list of characters.

Values

- When we pass (send) “Hello, world” to the println method, we are passing a value. This value, for this particular program is “Hello, world”.
- We can change that value to anything we need. It can be your name, a phone number, or an email address, depending on what the program is about.
- You should have tried that at home.
- Just like other programming languages, Java needs to know what kind of value you are working with.

Values

- Remember the ASCII table, each character, basically, can be represented with a single byte.
- A series of characters can be represented by a list, which we call a String, and we will discuss it in detail in the upcoming weeks.
- Let’s start with numbers, since even characters are represented by numbers.

Values

- Let’s think of mathematical functions.
- Addition operator, +, requires two values.
- 2 + 3
- Please notice that we are working with literal (real) values here.
- The computer needs to store these values somehow, and for integers, they generally use four bytes, or 32 bits.
- One of these bits, the most significant one (left-most), is used to denote the sign (either positive or negative).
- So, 31 bits are used to represent values, from -2,147,483,648 to 2,147,483,647.

Values

- Please notice, once again, that these are “values.”
- We have discussed 2 + 3, but for another operation it can be 120 + 387.
- So, when you want to print the result of this operation, we can write the following simple program.
- **As a reminder, please write down all these codes to practice. Even though it is possible to copy and paste, don’t do that.**

Values

Here is MathOperation.java file (notice that the public class name and the filename are the same).

```java
public class MathOperation {
  public static void main(String[] args) {
    System.out.println(2 + 3);
  }
}
```

We can perform mathematical operations and print them on the screen. So, we make use of your powerful processors to do job of a simple calculator.

Values

You can change 2 + 3 to 46 * 33, and it will print the result. We use * as the multiplication operator.

```
public class MathOperation {
  public static void main(String[] args) {
    System.out.println(46 * 33);
  }
}
```

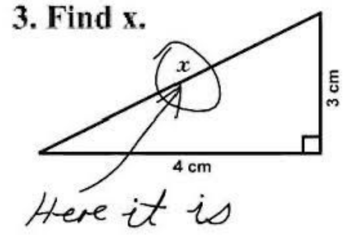
However, when we make this change, we have to compile the Java file again.

It is not fun to compile the complete program every time we make a change.

Variables

- Would it be possible to change these values while the program is running, and let the program calculate it with the values we enter.
- This is why we have programs. We run programs with different inputs.
  - We call the values we enter as inputs. Remember the programming diagram from last week, input → cpu → output.
- You can tell your web browser to go to google.com, or your favorite web site ieu.edu.tr - and these web addresses are different values for the web browser program.
- Then, what can we write instead of values? Variables.

Variables



- Variables, as the name suggests, can vary.
- They have some kinship to mathematical variables.
- For example, if $f(x) = x + 2$, then $f(3) = 3 + 2 = 5$. We use the value of 3 for the variable x.
- It is possible to **define** variables so that we can use them.
- As I have mentioned earlier, Java has to know what kind of variable you need.
- For integers, we can use the **int** type - basically the **int**eger we all know.

Variables

Here is the updated version of MathOperation.java.

```
public class MathOperation {
  public static void main(String[] args) {
    int a = 34;
    int b = 42;
    System.out.println(a + b);
  }
}
```

The “=” is not “equals,” but the “assignment” operator.

Variables

- The assignment operator **evaluates** its right hand side to find its “value” and “assigns” it to the “variable” on the left hand side.
- So,
  b = 42;
  is: the value of the right hand side is 42, we assign 42 to the variable **b**.
- Here is another:
  b = b + 4;
  This might look like a mathematical function where you can remove the variable **b**.
  No! In programming “=” is assignment.

Variables

- In
  b = b + 4;
  we evaluate the right hand side to find its “value”.
- Evaluation is done as follows:
- We notice that the statement “b + 4” has a variable. Each variable must be defined before we use them. So, it must have a value.
- We find the existing value of **b**, which is 42, and add 4 to it.
- Now “b + 4” has the “value” of 46.
- We assign the “value” of the right hand side to the left hand side.
- Once the statement is processed, variable **b** has the value of 46.

Variables

So, this code first defines variables **a** and **b**. To define a variable we tell the compiler their type, in this case **int**, and then we tell the variable name.

We can also set their initial value. Here we “assign” “values” 34 and 42 to them.

```
public class MathOperation {
  public static void main(String[] args) {
    int a = 34;
    int b = 42;
    System.out.println(a + b);
  }
}
```

Variables

The System.out line first finds the “value” of “a + b” and prints it on the screen.

```
public class MathOperation {
  public static void main(String[] args) {
    int a = 34;
    int b = 42;
    System.out.println(a + b);
  }
}
```

So, this is an improvement, but we still need to compile it every time we change the values for **int** variables **a** and **b**.

Back to “Getting Input”

- We can get input from various sources. In the very first days of your programming adventure, it is best if you get that input from your keyboard.
- To do so, I have to introduce System.out’s cousin: System.in.
- While System.out is about the output, System.in is about the input.
    - Remember, input → cpu → output.
- System.out is the terminal screen, we call it the “standard output.”
- System.in is then the “standard input” and it continuously reads anything you type with your keyboard.
    - The **input** → **cpu** → **output** becomes **keyboard** → **program** → **screen**.

Getting Input

- System.out.println saves us from a lot of trouble. It is an abstraction that handles the complicated procedure of printing something on the screen.
- Reading from the keyboard is also complicated, but Java provides tools to make this process a lot easier.
- To get input we use a **Scanner** object.
- We have to tell Java to load Scanner, because it is not loaded by default.
- We do this with the **import** statement **before** the class definition.
- Scanner's full name is java.util.Scanner.
    - This means there is a namespace (similar to directory) called *java*, in it there is another namespace called *util* (for utility), and in that we have a *Scanner* class.
    - Don't worry about the details for now.

Getting Input

- “Scanner” is the name of the class, which is the blueprint (remember last week) where its composition is defined.
- To get a Scanner object, we tell the compiler to create (construct) this object from the definition of the class, simply by

  ```
  Scanner sc = new Scanner(System.in);
  ```

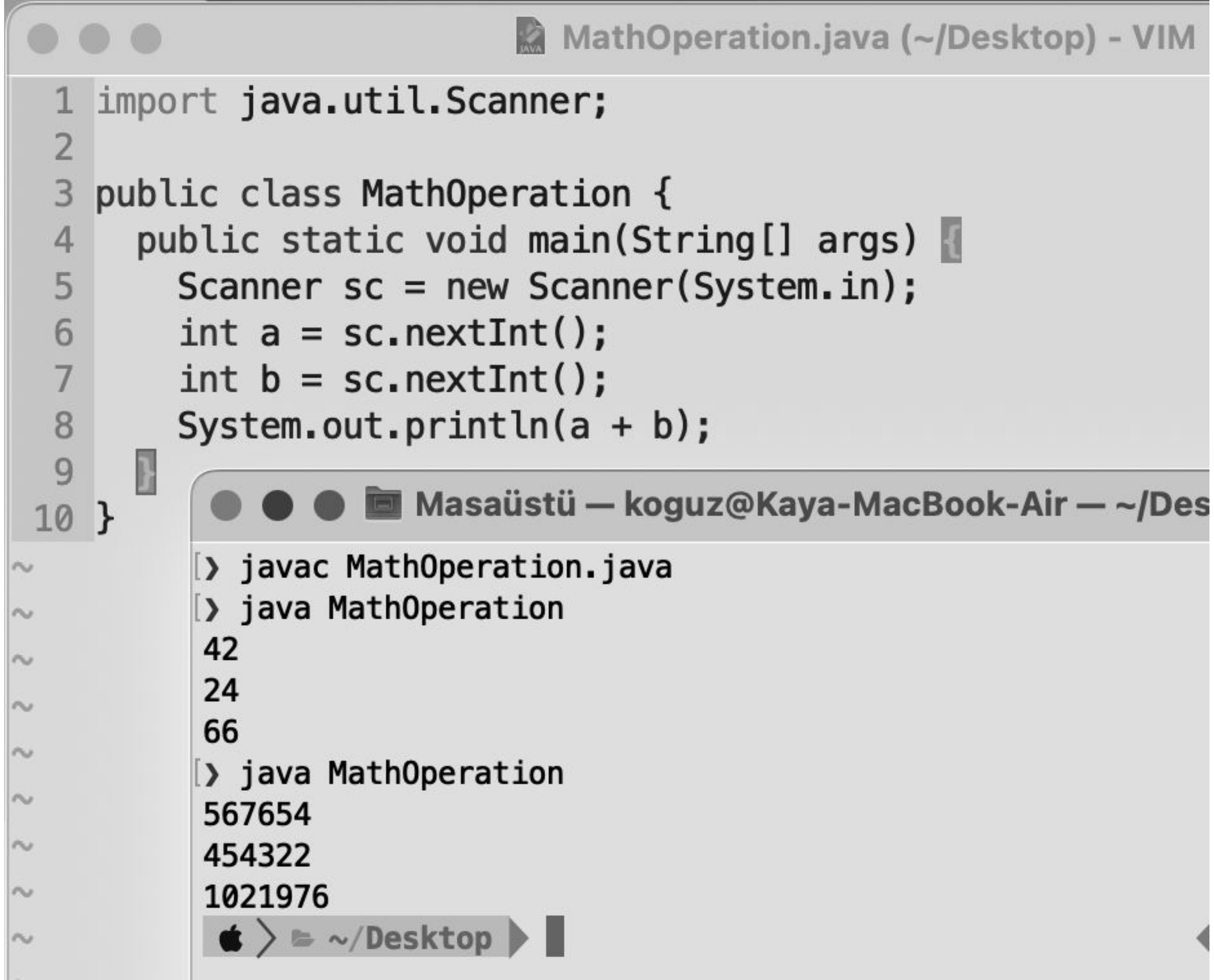
- Similar to “int a = 34,” “Scanner” on the left hand side is the type of the variable. “sc” is the name of the variable.
- Let’s discuss the right hand side, the “evaluation”, or finding the “value” that will be assigned.

Getting Input

- The right hand side uses the “new” **operator**. Notice that I use the term “operator”, such as + when we added two values: “**new**” does something.
- This “new” operator creates an instance (or object) of a class.
- Scanner class accepts some parameters: System.in.
- This tells that while creating (constructing) this object, use System.in.
- System.out and System.in are streams, the data flows through them.
- When I type on this keyboard, each character flows through standard input, and the browser reads it.
- When I use System.out.println, the value that I pass to it flows through standard output and they appear on the terminal screen.

Getting Input

- Basically, we construct a Scanner object that can read from the standard input, which is the keyboard. Fine.
- Similar to the println of System.out, System.in has **nextInt()** method which reads an integer from the user.
- Let’s use it.
- Let’s get some integer “value” and “assign” it to a “variable” and then print it on the screen with “println.”
- Exciting!



MathOperation.java (~/Desktop) - VIM
Masaüstü — koguz@Kaya-MacBook-Air — ~/Des

```java
import java.util.Scanner;

public class MathOperation {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int a = sc.nextInt();
    int b = sc.nextInt();
    System.out.println(a + b);
  }
}
```

```
❯ javac MathOperation.java
❯ java MathOperation
42
24
66
❯ java MathOperation
567654
454322
1021976
 ~/Desktop
```

Getting Input

- Now we are in business!
- We can get values, operate on them, and print results!
- We have to work much harder for more complicated operations, but all journeys start with these simple steps.
- Let’s go through the code.

Getting Input

Tell the compiler to load Scanner class.

```
import java.util.Scanner;

public class MathOperation {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int a = sc.nextInt();
    int b = sc.nextInt();
    System.out.println(a + b);
  }
}
```

Create a Scanner object, call it sc.

Get input from the user!

Add them up and print the result on screen!

Some technicalities

- Variable naming:
  - Java is case sensitive, so variable “sc” is not the same as “SC”.
  - The variable name MUST start with a character or underscore _ and may continue with them and additionally numerals. Some variable names:
    - score, _file, var1
- You should not use Java keywords or reserved words as variable names.
  - Do not use **int**, for example, as your variable name. Use **i**.
- Variable naming has a lot of conventions, I’ll talk about them when the time comes. For now, try to name them so that you can understand why they are being used when you see them.

Some technicalities

- Comments are vital. We can write comments that are ignored by the compiler, but includes information about a piece of code.
- Comments are written between /* */ when they span several lines.
- For a single line comment, we can use //.
- Do not comment the obvious, but use comments to help read the code.
- Make sure the comments are updated when the code is updated, too.

Java Primitives

- The following data types are called “primitives” in Java.
- We will use these types extensively because they represent the fundamental building blocks: numbers, characters, and logic.
- Visit https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html for more details.
  - Learn about two’s complement!

Java Primitives

byte	The byte data type is an 8-bit signed integer from -128 to 127.	**float**	The float data type is a single-precision 32-bit IEEE 754 floating point.
short	The short data type is a 16-bit signed integer from -32,768 to 32,767.	**double**	The double data type is a double-precision 64-bit IEEE 754 floating point.
int	The int data type is a 32-bit signed integer from -2^{31} to $2^{31}-1$.	**boolean**	The boolean data type has only two possible values: **true** and **false**.
long	The long data type is a 64-bit integer from -2^{63} to $2^{63}-1$.	**char**	The char data type is a single 16-bit Unicode character.

Java Primitives

- Unless you are having trouble with memory, use **int**.
- Unless you need a lot more precision with floating point numbers, use **float**.
- Always remember that **char** is a single character, not a **String**.
- Boolean logic is vital to programming, so use **boolean** when needed rather than an **int** value of 0 or 1.
- Notice that all primitives use lowercase. As convention Java classes start with an uppercase: Scanner, String to name a few.
- Primitives are NOT classes. They have values. Classes are something different. We will learn about them later.

Java Primitives

- Similar to Scanner.nextInt(), there are
  - nextByte, nextDouble, nextFloat to get other types.
- Take a look at Scanner description at Java documentation: https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/util/Scanner.html
- Always refer to Java Documentation rather than other websites.

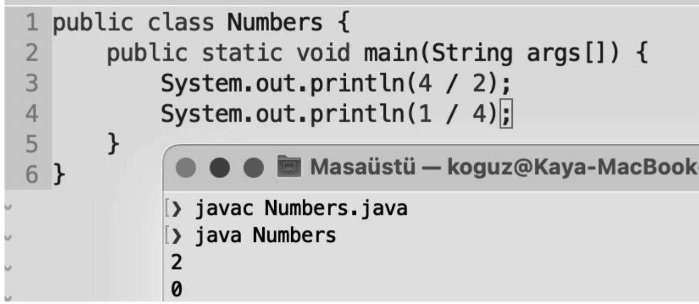
Numbers in Programming

- It is very easy to perform mathematical operations:
    - Use + for addition,
    - Use - for addition,
    - Use * for multiplication,
    - Use / for division,
    - Use % for remainder (modulus).
- To perform an operation on the same variable, use assignment operator as follows;
    - x += 5; // means x = x + 5;
    - x -= 2; // means x = x - 2;
    - x *= 3; // means x = x * 3;
    - x /= 4; // means x = x / 4;
    - x %= 2; // means x = x % 2;

Numbers in Programming

- There are also some unary operators;
  - Use ++ for increment (add 1 to value),
    - x++; // means x = x + 1;
  - Use -- for decrement (subtract 1 from value),
    - x--; // means x = x - 1;
  - Use ! for logical complement, turns true into false, and vice versa.
    - !x; // means true if x is false, false if x is true.
- The unary operators can be pre or post. The difference is vital. Let x=1, and consider preincrement and postincrement;
  - System.out.println(x++); // prints 1, but x now has the value of 2;
  - System.out.println(++x); // prints 2.

Numbers in Programming



```java
public class Numbers {
    public static void main(String args[]) {
        System.out.println(4 / 2);
        System.out.println(1 / 4);
    }
}
```

```
❯ javac Numbers.java
❯ java Numbers
2
0
```

- What is the value of 1 divided by 4?
- It is 0.25, or ¼, or a quarter… These are all representations that we use.
- What is the value of 4 / 2?
- It is 2, but 4/2 also represents 2.
- The way we **interpret** these values are not the same as the programming language.
- For example, if you divide two integers, Java returns the result as an integer.
- For 4/2, it is not a problem, but for ¼ Java returns 0, since it cannot represent 0.25 as an integer.

Numbers in Programming

- The division operation has the possibility of creating floating point numbers.
- This is very common since we use “average” values for a lot of scenarios, such as the “average score of midterm scores of everyone in this class.”
- We can store all your scores as **float** variables, or we can **cast** them to floats when we are dividing them.
- Casting for me will always be “casting a magic spell” but it is also used in the “casting of roles for a movie or theatre play.”
- For the casting of a movie, some actor plays or acts the role of a character.
- So we tell Java to cast an integer into a float, which it does, the integer is represented as a float.

Numbers in Programming

- Float is more precise, so when we cast an integer to a float its value is not lost, 3 becomes 3.0.
- But if you cast a float to an integer, then you will lose precision, 3.14 will become 3!
  - Did you get the “take pi (π) as 3” joke?
- To cast, simply write the type you want to cast to in parenthesis.
- (float) 4; // 4 becomes 4.0
- (int) pi; // variable pi is cast to int.
- Let’s try it.

Numbers in Programming

```
/* This is a multi line comment.
   September 3rd, 2022
   SE115 Examples */

public class Cast {
    public static void main(String args[]) {
        float pi = 3.14f;  // Value of PI
        int r = 2;
        System.out.println("Pi is " + pi + ", radius is " + r);
        System.out.println("Area is " + (pi*r*r));
        System.out.println("Taking pi as 3: " + ((int)pi*r*r));
    }
}
```

f is required to define a float:
float x = 3.0f;

casts pi to int:
(int)pi

System.out.println()

- We can use println to print the “values” of variables.

```
public class Printing {
    public static void main(String args[]) {
        int a = 12;
        int b = 42;
        System.out.println("Sum of " + a + " and " + b + " is " + (a+b));

        float pi = 3.14f;
        System.out.println("Value of pi is " + pi + " but you can use " + (int)pi);
    }
}
```

Parentheses here are very important, it tells Java to evaluate the expression and print its value.

printf

- The original **printf** is a C function.
- The name is “print formatted” (not **f**ile).
- We can format the output if we want to, for example, we can print 1 as 01, or 3.343242 as 3.34, using only two digits after the point.
- For Java, printf is under the Formatter class, and it can also be used to format dates.
- https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/util/Formatter.html

printf

- We use conversion characters that correspond to variables or values to print formatted lines.
- %s for Strings, %d for decimals (integers), %f for floats.
- We can use numbers to denote the number of digits. Here is an example:

- `System.out.printf("Hello %s, %d, %02d, %.3f\n", "SE 115", 13, 1, 2.12345);`

which prints

```
Hello SE 115, 13, 01, 2,123
```

Escaping

- What if you want to print something that has quotes?
- System.out.printf("He said "hello" to me");
- Immediately you will notice that the syntax coloring changes.
- We can “escape” these characters with a backslash:
- System.out.printf("He said \"hello\" to me");
- So that they are not interpreted in a special way, but as literal characters.
- You can escape backslash with another backslash: \\
- \n represent a special character that returns a new line.
- System.out.printf("He said \"hello\" to me\n");

Next Week

Flow Control