Lecture 5

Math

- In mathematics, you have probably seen functions like sin and log, and you have learned to evaluate expressions like sin($\pi/2$) and log($1/x$). First, you evaluate the expression in parentheses, which is called the **argument** of the function. Then you can evaluate the function itself, maybe by punching it into a calculator.

- The Java library includes a **Math** class that provides common mathematical operations. **Math** is in the **java.lang** package, so you don't have to import it.

Math

- The Math.max(x,y) method can be used to find the highest value of x and y

  ```
  Math.max(5, 10);
  ```

- The Math.min(x,y) method can be used to find the lowest value of x and y:

  ```
  Math.min(5, 10);
  ```

- The Math.sqrt(x) method returns the square root of x:

  ```
  Math.sqrt(64);
  ```

- The Math.pow(x,y) method returns the x power y:

  ```
  Math.pow(2,3);
  ```

Math

- The Math.abs(x) method returns the absolute (positive) value of x:

```
Math.abs(-4.7);
```

- Math.random() returns a random number between 0.0 (inclusive), and 1.0 (exclusive):

```
Math.random();
```

- To get more control over the random number, e.g. you only want a random number between 0 and 100, you can use the following formula:

```
int randomNum = (int)(Math.random() * 101);  // 0 to 100
```

Exercise

- Ex1: Write prime test, and test if 12345678912345682**3** is prime or not.

- Ex2: Write an optimized prime test, and test if 12345678912345682**3** is prime or not.

Exercise

- Exercise 3: Take a user input number, print out the square root of this number.

- Exercise 4: Take a user input number, print out the forth root of this number.

- Exercise 5: Take two number n1, n2 input from user, print out the $n1^{n2}$.

- Exercise 6: Take a user input number, print out the cube root of this number.

Method / Function

- A **method** is a block of code which only runs when it is called.

- You can pass data, known as parameters, into a method.

- Methods are used to perform certain actions, and they are also known as **functions**.

- Why use methods? To reuse code: define the code once, and use it many times.

Create a Method

- A method must be declared within a class. It is defined with the name of the method, followed by parentheses **()**. Java provides some pre-defined methods, such as System.out.println(), but you can also create your own methods to perform certain actions:

```
public class Main {
    static void myMethod() {
        // code to be executed
    }
}
```

- **static** means that the method belongs to the Main class and not an object of the Main class.

Parameters and Arguments

- Information can be passed to methods as parameter. Parameters act as variables inside the method.

- Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

- The following example has a method that takes a String called **fname** as parameter. When the method is called, we pass along a first name, which is used inside the method to print the full name:

Parameters and Arguments

- When a **parameter** is passed to the method, it is called an **argument**.

```
public class Main {
  static void myMethod(String fname) {
      System.out.println(“Hello” + fname);
  }
  public static void main(String[] args) {
      myMethod(“Liam");
      myMethod(“Jenny”);
      myMethod(“Anja");
  }
}
// Liam Refsnes
// Jenny Refsnes
// Anja Refsnes
```

Multiple Parameters

- When you are working with multiple parameters, the method call must have the same number of arguments as there are parameters, and the arguments must be passed in the same order.

```
public class Main {
    static void myMethod(String fname, int age) {
        System.out.println(fname + " is " + age);
    }
    public static void main(String[] args) {
        myMethod("Liam", 5);
        myMethod("Jenny", 8);
        myMethod("Anja", 31);
    }
}
// Liam is 5
// Jenny is 8
// Anja is 31
```

Return Values

- The **void** keyword, used in the examples above, indicates that the method should not return a value. If you want the method to return a value, you can use a primitive data type (such as int, char, etc.) instead of void, and use the **return** keyword inside the method

Method Overloading

- Within one class, you can have two (or more) definitions of a single method name. This is called **overloading** the method name.
- When you overload a method name, any two definitions of the method name must have different signatures; that is, any two definitions of the method name either must have different numbers of parameters or some parameter position must be of differing types in the two definitions.

Method Overloading

- With **method overloading**, multiple methods can have the same name with different parameters

```
int myMethod(int x);
float myMethod(float x);
double myMethod(double x, double y);
```

Overloading and Automatic Type Conversion

- The **signature** of a method consists of the method name and the list of types for parameters that are listed in the heading of the method name.

- Java always looks for a method signature that exactly matches the method invocation before it tries to use **automatic type conversion**.

- If Java can find a definition of a method that exactly matches the types of the arguments, it uses that definition.

- Only after it fails to find an exact match does Java try automatic type conversions to find a method definition that matches the (type cast) types of the method invocation.

You Cannot Overload Operators in Java

- Many programming languages, such as C++, allow you to overload an operator, such as +, so that the operator can be used with objects of some class you define as well as be used for such things as numbers.

- You cannot do this in Java. If you want to have an “addition” in your class, you must use a method name, such as add, and ordinary method syntax; you cannot define operators, such as the + operator, to work with objects of a class you define.

Method Scope

- Variables declared directly inside a method are available anywhere in the method following the line of code in which they were declared.

```
public class Main {
    public static void main(String[] args) {
        // Code here CANNOT use x
        int x = 100;
        // Code here can use x
        System.out.println(x);
    }
}
```

Block Scope

- A block of code refers to all of the code between curly braces {}.
- Variables declared inside blocks of code are only accessible by the code between the curly braces, which follows the line in which the variable was declared

```
public class Main {
    public static void main(String[] args) {
        // Code here CANNOT use x
        {
            // This is a block
            // Code here CANNOT use x
            int x = 100;
            // Code here CAN use x
            System.out.println(x);
        } // The block ends here
        // Code here CANNOT use x
    }
}
```

Exercise

- Exercise 7: Write a function called printHello() that outputs “Hello” on the screen.

- Exercise 8: Write a function called circleArea() that takes an double value from parameter, calculates the area of the circle, returns the area of the circle (pi=3.14, circle area = pi * r^2)

- Exercise 9: Write a function called displayReverse() that takes an integer and displays the reverse of that number on the screen.

- Exercise 10: Write a function called numberToTens() that takes an integer and returns the corresponding number in one and zeros. For example: if the number is 54321, returns 10000.

Exercise

- Exercise 11. Write a program that defines and tests a factorial function. The factorial of a number is the product of all whole numbers from 1 to N. For example, the factorial of 5 is 1 * 2 * 3 * 4 * 5 = 120

- Exercise 12. Write a program that finds the first prime number greater than 1 billion (1,000,000,000).

Exercise

- Exercise 13: User enter A,B,C. Apply the quadratic formula to find solution.

- Exercise 14: Write a program that simulate roll the dice ten times, each round should output random number 1~6.

Exercise

- Exercise 15: Write function to calculate the triangle for given number. Ex: triangle(5) =1+2+3+4+5=15

- Exercise 16: Write function to find the smallest factor of given number. (The smallest factor exclude 1, Ex: Smallest(15) => 3, Smallest(7) => 7)

Exercise

- Exercise 17: Write a function named print_out that prints all the whole numbers from 1 to N. Test the function by placing it in a program that passes a number n to print_out, where this number is entered from the keyboard. The print_out function should have type void; it does not return a value. The function can be called with a simple statement: print_out(n);

- Exercise 18: Write a function do primetest. Optimize the prime-number function by calculating the square root of n only once during each function call. Declare a global variable sqrt_of_n of type double. (Hint: A variable is local if it is declared inside the function.) Then use this variable in the loop condition, test case: 12345678912345682 3.

Exercise

Exercise 19: Write a function multiPrint(int n, char c) that prints n copies of a character c. Apply the function in program to print triangles, upside down triangles, and diamonds.

Exercise 20: Write a function isPrime to test whether a parameter is prime. Apply the function in a program which prints all the prime numbers up to 100.

Exercise

Exercise 21: Write a function reverse(int n) which reverses the digits in its parameter and return the result. For example, if n is 927, it would return 729. Apply the function in a program that asks the user to enter 10 numbers and reverse them.

Exercise 22: Write a function quadratic(int a, int b, int c, double x) which calculates the value of the quadratic: $ax^2 + bx + c$

Exercise 23: Write a function factorIt that writes out the prime factorization of an integer parameter.

Arrays and Functions

- Just like regular variables, arrays can be passed into functions.

- When passing arrays into functions, we should consider this first:

  - Pass the **entire arrays** into the sub function

    OR

  - Pass just **one element of array** into the sub function.

Array Indexed Variables as Arguments

- An array indexed variable can be used as an argument anyplace that a variable of the array's base type can be used. For example, suppose you have the following:

- double[] a = new double[10];

- Indexed variables such as **a[3]** and **a[index]** can then be used as arguments to any method that accepts a **double** as an argument.

Entire Array as Arguments

- You can also define a method that has a formal parameter for an entire array so that when the method is called, the argument that is plugged in for this formal parameter is an entire array.

- Whenever you need to specify an array type, the type name has the form Base_Type[], so this is how you specify a parameter type for an entire array.

Entire Array as Arguments

- An array type is a reference type just as a class type is, so, as with a class type argument, a method can change the data in an array argument. To phrase it more precisely, a method can change the values stored in the indexed variables of an array argument.

- An array type parameter does not specify the length of the array argument that may be plugged in for the parameter. An array knows its length and stores it in the length instance variable. The same array parameter can be replaced with array arguments of different lengths.

Exercise

- Exercise 24: Write a program that initializes array with 3,4,5,2,3. Write a function call minArray which accepts 1D array, and return the minimum of the array.

- Exercise 25: Write a program that initializes array with 3,4,5,2,3. Write a function call avgArray which accepts 1D array, and return the average of the array.

Exercise

- Exercise 26: Write a program that initializes 2D array with {{3,4,5,2,3}, {2,3,4,5,1}, {6,5,4,3,2}}. Write a function call minArray which accepts 2D array, and return the minimum of the 2D array.

- Exercise 27: Write a program that initializes array with {{3,4,5,2,3}, {2,3,4,5,1}, {6,5,4,3,2}}. Write a function call avgArray which accepts 2D array, and return the average of the 2D array.

Exercise

- Exercise 28: Write a complete program that does the following.
  1. It asks the user to enter 9 integers as the entries of a 3 x 3 table.
  2. The program reads the 9 entries, row by row and prints the table.
  3. If every row and column of the table have the same sum then the program adds the message: MAGIC.

  Here is an example of how the program should work:

  Enter 9 entries of a 3 x 3 table: **10 14 18 15 16 11 17 12 13**
  output:
  10 14 18
  15 16 11
  17 12 13
  MAGIC