Object-Orientated Programming

CMM024

GOLOVINE (2023)

Session 2 (Procedural Programming)

Introduction to:

- Conditional statements
- For loops
- Conditional loops

CMM024

Summary

- **A few facts about variables**
- **How to format output (“printf”)**
- **Using in built functions (“Math….”)**
- **Brackets and scope ({, })**
- **Condition statement (if, switch(…){case ….})**
- **For loops & relational operators (for(…){})**
- **Conditional loops (while(){} ; do{}while(…))**
- **Next Session**

Facts about variable type casting

- A lot of compile errors comes from assigning the wrong value to a variable. We have seen that Java will stop you from converting value from one type to another is it ends up loosing precision. A good example is float to double.
- TypeCasting uses “(type)”, where the type can a related type to original type. I.e. you cannot type cast a String to a double!
- For decimal values, the default type is double
- If you want to assign a decimal value to a float use F at the end

```
float intFloat = 76.32F;
```

- For boolean values, the default is false
- For whole number values, the default is integer

```
public class Lecture2_2 {

    public static void main(String[] args){

        double d = 575.7 / 34.7;
        float f = (float)d;
        int i = (int)d;
        System.out.println(
       “\nd: " + d +
       " \nf: " + f +
       " \ni: " + i + "\n");

    }
}
```

```
d: 16.59077809798271
f: 16.590778
i: 16
```


LET’S CODE

Remember float precision is 6 to 7 digits

Tests

Expression	Result TYPE	Implications	Implications
6.48/5	double	1.296	One is double and one is int hence result is double
7/ 9.99	double	1.42714285714286	One is int and one is double (not float: 9.99F) hence double
6/8	int	0	Both are int so result int -> truncation
2.99/1.27	double	2.35433070866142	Both are double so result is double
9F- 2.12345	double	7.0	One is float and one is int -> default will be double (standard)
7.6 / 6.26F	double	1.21405746359813	One is double and one is float hence result is double
(int)(23.9 * 4.21)	int	100	Regardless what is inside bracket it is typecast to int -> truncation
(int)(5.9F * 4.21)	int	24	Regardless what is inside bracket it is typecast to int -> truncation
(float)(1.23 - 7.1f)	float	-5.9	One is double and one is float hence result is double. Overall 6 decimal digit precision Note that one has two decimal digits and other 1. Hence result will be 2 decimal digits
(int)(6.7F/5.4) - (1.23 - 6.5F)	What will be the result type and how many decimal digits are you expecting to have in the prinltn statement?		

Using printf for more accuracy

- System.out.println() decides how to represent values displayed. YOU HAVE NO CONTROL!
- A better way to display these values is to use System.out.printf() method as it gives YOU THAT CONTROL

```
public class Lecture2_3 {
    public static void main(String[] args) {
        float d_E_P_b = 5.0578F;
        double d_E_P_km = d_E_P_b * 1000000000;
        int c_km = 299792458 / 1000;
        double tL_E_P = d_E_P_km / c_km;
        System.out.println(
         “\nTime taken for light emitted from Earth to reach Pluto: "
            + tL_E_P + " (seconds)\n”);
    }
}
```

Do we need all these digits after the seconds?

Speed of light is 299,792,458 meters per second

Distance to Pluto from Earth is 5.0578 billions km

How long the light emitted from earth to reach Pluto?

```
/jdt_ws/Lecture2_67beeac3/bin Lecture2_3

Time taken for light emitted from Earth to reach PLuto: 16871.02951379623 (seconds)
```


LET’S CODE

Using printf for more accuracy

- In the previous slide we used “ System.out.println”. This does not allow to format anything. Java has also the “System.out.printf” derived from the C++ era. This can be used for all types of variables. Here we use our constants

- We use “%8.4f” for the decimal result. This means that we will have only 4 digits for the decimal part from a total of 8 digits. Spaces will pad the numbers when necessary!

Format	Type	Example
D	Decimal integer	1234
F	Fixed floating point	1234.990
E	Exponential floating point	1.234E+04
G	General floating point (`f` or `e` depending on the size of the value)	1234.0
S	String	Hello World

```
public class Lecture2_3 {
    public static void main(String[] args) {
        float d_E_P_b = 5.0578F;
        double d_E_P_km = d_E_P_b * 1000000000;
        int c_km = 299792458 / 1000;
        double tL_E_P = d_E_P_km / c_km;
        System.out.println("\nTime taken for light emitted from Earth to reach PLuto: "
            + tL_E_P + " (seconds)\n");
        System.out.printf(
                "Time taken for light emitted from Earth to reach Pluto:\n%6.0f (seconds) \n %8.4f (hours)\n",
                tL_E_P, tL_E_P / 3600 );
    }
}
```

```
Time taken for light emitted from Earth to reach PLuto: 16871.02951379623 (seconds)

Time taken for light emitted from Earth to reach PLuto:
   16871 (seconds)
   4.6864 (hours)
```


LET’S CODE

Constants & using printf to display results

- We used variables to hold values in previous example. However the speed of light will never change and so is the average distance of Pluto from Earth.
    - We can use constants to hold these values instead!
    - The CPU handles constants much faster!

```
public class Lecture2_3 {
    public static void main(String[] args) {
        final float d_E_P = 5.0578F * 1000000000;
        final int c_km = 299792458 / 1000;
        final double tL_E_P = d_E_P / c_km;
        System.out.println("\nTime taken for light emitted from Earth to reach PLuto: "
            + tL_E_P + " (seconds)\n");
        System.out.printf(
            "Time taken for light emitted from Earth to reach PLuto:\n%8.0f (seconds) \n %8.4f (hours)\n\n",
            tL_E_P, tL_E_P / 3600 );
    }
}
```

```
Time taken for light emitted from Earth to reach PLuto: 16871.02951379623 (seconds)

Time taken for light emitted from Earth to reach PLuto:
   16871 (seconds)
   4.6864 (hours)
```

Constants & use printf to display results

- Variables that are often use in programming are constants. Once initialised with a value they cannot be changed. Hence the name of constant. We use the keyword final to make a variable immutable
- Like any variable, a constant can be used in any calculation. How many miles a photon will travel in 1 minute?
- Let's create a new java file and call it ConstantDemo.java, and create a main methods.

```
public static void main(String[] args) {
        // light speed (meters/second) usually called c
        final int c = 299792458;
        // light speed (km/second) needed because distances in kms
        final float c_km = c / 1000;
        // distance from Earth to Mars (millions kms)
        final float dist_e_m = 54.6F;
        // distance from Earth to Pluto (billions km)
        final float dist_e_p = 5.0579F;
        // distance from Earth to Mars (kms)
        final float dist_e_m_km = dist_e_m * 1000000;
        // distance from Earth to Pluto (kms)
        final float dist_e_p_km = dist_e_p * 1000000000;
        System.out.println("\nTime for light emitted from Earth to reach these planets");
        System.out.println("Planets\tDistance\t      Seconds\t       Minutes\t       Hours");
        // time for light emited from earth to reach Mars in kilometers in seconds
        final double t_m_s = dist_e_m_km / c_km;
        System.out.printf("Mars\t%.0f\t%10.2f(s)\t%10.4f(mn)\t%8.4f(hr)\n", dist_e_m_km,t_m_s,t_m_s/60, t_m_s/3600);
        final double t_p_s = dist_e_p_km / c_km;
        System.out.printf("Pluto\t%.0f\t%10.2f(s)\t%10.4f(mn)\t%8.4f(hr)\n", dist_e_p_km, t_p_s, t_p_s/60, t_p_s/3600);
    }
```

```
Time for light emitted from Earth to reach these planets
PLanets Distance              Seconds         Minutes         Hours
Mars    54600000            182.13(s)       3.0354(mn)    0.0506(hr)
Pluto   5057900032        16871.36(s)     281.1894(mn)    4.6865(hr)
```


THIS IS AN EXAMPLE!

Conditional statement (IF)

- Often we must compare a value with another one.
- In Java we use the keyword “if”
  - We use for the condition >, <, ==, <= , >= or !=
    - Bigger, smaller, equal, bigger or equal, smaller or equal or not equal
  - You can have multiple condition by using && or ||
    - AND or OR
- Example: we want to test a value that can range from 0 to 100 to check if bigger or smaller or equal to 50 (1)
- Check if eligible to donate blood. Must be 17 or over and weigh more than 50 kg (2)
- What about if the person age has to be less than 65? (3)

```
import java.util.Scanner;

public class Checking {

    public static void main(String[] args) {
        System.out.println("Enter a number please");
        Scanner input = new Scanner(System.in);
        int value = input.nextInt();
        if (value > 50) {
            System.out.println("Value bigger than 50");
        } else if (value < 50) {
            System.out.println("Value smaller than 50");
        } else {
            System.out.println("Value is equal to 50");
        }
        input.close();
    }

}
```

```
public static void main(String[] args) {
  Scanner input = new Scanner(System.in);
  System.out.println("Enter your age");
  int age = input.nextInt();
  System.out.println("Enter your weight (Kg)");
  int weight = input.nextInt();
   if (age >= 17 && age < 65 && weight > 50) {
       System.out.println("You are eligible to donate
blood");
    } else{
       System.out.println("Sorry you cannot donate
blood");
     }
     input.close();
    }
```

LET’S CODE

Other Conditional statement

- Unlike if-then and if-then-else statements, the switch statement can have a number of possible execution paths
- Write some code to display the name for months. This can be laborious when using if and else statements!

```
int month = 8;
if (month == 1) {
    System.out.println("January");
} else if (month == 2) {
    System.out.println("February");
}
... // and so on
```

- A better solution is using the switch statement
  - It is more structured and readable

```
int month = 8;
String monthString;
switch (month) {
  case 1:  monthString = “January";
           break;
  case 2:  monthString = "February";
           break;
  case 3:  monthString = "March";
           break;
  case 4:  monthString = "April";
           break;
  case 5:  monthString = “May";
           break;
  case 6:  monthString = "June";
           break;
  case 7:  monthString = "July";
           break;
  case 8:  monthString = "August";
           break;
  case 9:  monthString = "September";
           break;
  case 10: monthString = "October";
           break;
  case 11: monthString = "November";
           break;
  case 12: monthString = "December";
           break;
  default: monthString = "Invalid month";
           break;
}
System.out.println(monthString);
```

Loops: for

For loops are very useful to repeat some code statements a number of times that is predetermined

To create a loop, you must have a variable that is initially initialised. It increases or decreases each time one loop complete. After a certain amount of times the for loop stops when the variable reaches a certain value

The more common form of the for loop is to declare a variable and initialise it at the same time “int t = 0;”.

Curly brackets “{}” are used to determine the scope of the code that will run (here: System.out.println(“t = “ + t);) each time the loop runs.

The condition clause controls when the loop ends. Here we tell the loop to keep running while t is smaller than 10 “t < 10;”. Once it reaches 10, the *for* loop stops.

```
//Typical loop

for ( <var initialisation>; <condition>; <increment>)
{
    //write the code that will be excused a finite number of times
}
```

```
public class ForLoopDemo {
    public static void main(String[] args){
        for ( int t = 0; t < 5; t = t + 1) {
            System.out.println("t= " + t);
        }
    }
}
```

```
p /Users/jean-c
/jdt_ws/Lecture2
t= 0
t= 1
t= 2
t= 3
t= 4
jean-claude@Mac
```

LET’S CODE

For loop (increment/decrement)

(Statement1: int i = 0;) sets and initialise a variable before the loop starts

(Statement 2: i < 10;) defines the condition for the loop to run (i must be less than 10). If the condition is true, the loop will continue, if it is false, the loop will end

(Statement 3: i = i + 1) increases a value of i each time the code block inside the {} in the loop is executed. Note if this did not exists, the loop would run for ever

Each time the loop completes a cycle, the increment i either increases or decreases i = i + 1 / i = i - 1 . This increment directly controls the for loop. By changing the increment we change the number of times the loop run before ending.

Other way to increment or decrement I + + / i - -

```java
public static void main(String[] args) {

    for (int i = 10; i > 0; i = i - 1) {

        System.out.println("Loop: " + i);

    }

}
```

```
Loop: 10
Loop: 9
Loop: 8
Loop: 7
Loop: 6
Loop: 5
Loop: 4
Loop: 3
Loop: 2
Loop: 1
```

```java
public static void main(String[] args) {

    for (int i = 0; i < 10; i = i + 1) {

        System.out.println("Loop: " + i);

    }

}
```

```
Loop: 0
Loop: 1
Loop: 2
Loop: 3
Loop: 4
Loop: 5
Loop: 6
Loop: 7
Loop: 8
Loop: 9
```

```java
public static void main(String[] args) {

    for (int i = 0; i < 10; i = i + 2) {

        System.out.println("Loop: " + i);

    }

}
```

```
Loop: 0
Loop: 2
Loop: 4
Loop: 6
Loop: 8
```

Loop examples

For loop	Result output	Note
for (int j = 5; j >= 0; j--)	5,4,3,2,1,0	No problems. Note: because of >= 0 is also included
for (int i = 0; i < 5; i++)	0,1,2,3,4	No problems
for (int j = 0; j >= 0; j--)	0	0 is fine because >=, but then j become negative thus loop stops
for (int t = 0; t != 20; t += 2)	0,2,4,6,8,10,12,14,16,18	The loop stops at 18 because 20 will break t != 20 (different)
for (int u = 1; u > 5; u += 2)	NOTHING!	The condition states that u must be bigger than 5. We start with u = 1, thus the loop breaks before starting!
for (int k = 0; k != 5; k += 2)	Infinite loop	k starts with 0, then increased by 2 each time the loop runs. Thus k is always an even number. However the condition looks a odd number. The loop will never terminate

Other kind of loops

- We have looked at the for loop when we do know how many times the loop must repeat some code to run. There are many situations when we do not know when we must stop the loops
    - A menu when user must press a specific key to terminate a program.
    - Games when we cannot expect how long the main loop must run
- In the “for" loop we used a variable that is decreased or increased and when running that value is changed and when it reaches a specific value, the loop terminates. There are other loops that allow us more liberty to terminate a loop, check if the condition is right before ended the loop.
    - While loop
    - do-while loop
- Both have a condition that must be met to terminate a loop. However the while loop checks the condition at the beginning while the do-while check it at the end

While loop

- Loops can execute a block of code as long as a specified condition is reached. We have seen the FOR loop that does run a predetermined number of times.
- The WHILE loop is different as it runs until a specified condition is TRUE

We must set the condition before

Then we check if the loop keeps on going

If it does we do something

Now we check if the condition is still true

```
boolean condition = true;

While (condition is true){
    Do something….
    Is condition remains true?
}
```

If it does the loop repeat itself

While loop in action

This is a simple dice guessing game.

- First we get a random number from 1 to 6.
  - Remember random range from 0.0 to 1.0, where 1 is excluded.
  - Without + 1 we would have 0 to 5
  - With + 1 we have 1 to 6
- We create a while loop that will terminate when user guess is equal to the random number by setting continueGuessing = false
  - You must remember to set continueGuessing = true before the while loop! Why?
- So you get a guess using the Scanner and compare that guess to the random number
  - If equal we display a congratulation message and set continueGuessing to false, which will stop the loop
  - If not equal, display the message try again, and repeat the loop!
- Note: if you are very unlucky you can spend some time before your guess is right.

```
mport java.util.Scanner;

public class WhileLoops {
    public static void main(String[] args) {

        //it is a dice guessing game so from 1 to 6.
        //so here from 0.00000 to 5.9999999
        //casted to int hence 0 to 5 as no decimal
        //then we add 1 so 1 to 6
        int correct_guess = (int)(Math.random() * 6) + 1;
        int guess;
        boolean continueGuessing = true;
        Scanner input = new Scanner(System.in);
        while (continueGuessing) {
            System.out.println("Guess a number");
            guess = input.nextInt();
            if (guess == correct_guess) {
                System.out.println("Congratulations! You won!");
                continueGuessing = false;
            } else {
                System.out.println("Try again");
            }
        }
    }
}
```

```
Try again
Guess a number
6
Try again
Guess a number
4
Try again
Guess a number
3
Congratulations! You won!
jean-claude@MacBook-Pro-2 CMM024_Lectures %
```

LET'S CODE

While loop in action

This is a more complex version of the dice guessing game. You have 3 throws to find the number only. If you pass that number you have lost.

- First we get a random number from 1 to 6.
- We set a limit to the number of throws. Here 3 guess_limit = 3; and we must initialise guess_count to 1. WHY?
- We create a while loop that will terminate when user guess is equal to the random number by setting continueGuessing = false. We assume it is true to begin with: continueGuessing = true; WHY?
- If you have passed the throw limit, you display a message "You have lost" and terminate the while loop
- If you still have throws, then they are 2 possibilities.
- If the guess is right.
  - Display a congratulation message and set continueGuessing to false, which will stop the loop
- If your guess is wrong
  - Only display a message "your guess is wrong: try again"
- We increase the guess count by one....(guess_count += 1;)

```java
import java.util.Scanner;
public class GuessingGame3 {
    public static void main(String[] args) {
        int correct_guess = (int) (Math.random() * 6) + 1;
        int guess_count = 1;
        int guess_limit = 3;
        int guess;
        boolean continueGuessing = true;
        Scanner input = new Scanner(System.in);
        while (continueGuessing) {
            System.out.println("Guess a number");
            guess = input.nextInt();
            if (guess_count > guess_limit) {
                System.out.println("You have lost the game");
                continueGuessing = false;
            } else {
                System.out.println("Guess a number");
                guess = input.nextInt();
                if (guess == correct_guess) {
                    System.out.println("Congratulations! You won!");
                    continueGuessing = false;
                } else {
                    System.out.println("Wrong guess: try again");
                }
            }
            guess_count += 1;
        }
        input.close();
    }
}
```

```
Enter dice number (0 to 6)
1
`try again
Enter dice number (0 to 6)
2
`try again
Enter dice number (0 to 6)
2
`try again
Max guess exeeded. You have lost
```

LET'S CODE

While loop in action

This is a more complex version of the dice guessing game using the do while loop

- Because we only check continueGuessing at the end, the loop will at least run once
- We do not need to set the condition to true to begin with
  - We need to be careful to avoid infinite loops!
  - Hence, this is often used for menus or situation when the loop termination is defined inside the loop itself.

```
public static void main(String[] args){
  int correct_guess = (int) (Math.random() * 6) + 1;
  int guess_count = 0;
  int guess_limit = 3;
  int guess;
  boolean continueGuessing;
  Scanner input = new Scanner(System.in);
  do {
      System.out.println("Guess a number");
      guess = input.nextInt();
      guess_count += 1;
      //have passed the total number of tries?
      if (guess_count >= guess_limit) {
         //YES! Stop the loop
         System.out.println("You have lost the game");
         continueGuessing = false;
      } else {
        //NOT YET
        if (guess == correct_guess) {
          //if the guess is correct you win...! Stop the loop
           System.out.println("Congratulations! You won!");
           continueGuessing = false;
        } else {
          //if the guess was wrong... continue the loop
          System.out.println("Wrong guess: try again");
          continueGuessing = true;
        }
      }
  } while(continueGuessing);
    input.close();
}
```

Next week

- **What are methods**
- **Creating your own methods**
- **How to invoke your own methods**
- **Method return value**