

Basic Programming

Flow Control and Loops in Java



Q1. Which is not a primitive data type?

- A. int
- B. double
- C. char
- D. String

Q2. How to declare a int variable?

- A. `int i = 5;`
- B. `int i == 5;`
- C. `int i = "5";`

Q3. What computer use for data encoding?

- A. Parts**
- B. Bits**
- C. Elements**
- D. Dots**

Q4. What is the highest value possible to encode on unsigned 8 bits?

- A. 127
- B. 255
- C. 300

Q5. How many bits int has?

- A. 8
- B. 64
- C. 32
- D. 16

Q6. How to declare a float variable?

- A. `float i = 2.5;`
- B. `float i = 2,50;`
- C. `float i = 2.5f;`

Q7. What is a String?

- A. Object type
- B. Primitive type
- C. Flow type

Q8. Which calculation gives 'true' result?

```
int i = 5;
```

```
int z = 9
```

- A. `(i < z) || (i == z)`
- B. `(z > i) && (i * z < 45)`
- C. `false || true && (z == i)`

Q9. Which is correct?

```
int a = 4
```

```
A. if(a%3==0){  
    System.out.println(a+" is divisible by 3");  
}  
other{  
    System.out.println(a+" is not divisible by 3");  
}  
B. if(a%3==0){  
    System.out.println(a+" is divisible by 3");  
}  
else{  
    System.out.println(a+" is not divisible by 3");  
}  
C. which(a%3==0){  
    System.out.println(a+" is divisible by 3");  
}
```

Q. Answers

1. D
2. A
3. B
4. B
5. C
6. C
7. A
8. A
9. B

Quick repetition... Lesson_2: “If we have logic...”

```
if(a%3==0){
```

```
System.out.println(a+“ is divisible by 3”);
```

```
} else {
```

```
System.out.println(a+“ is not divisible by 3”);
```

```
}
```

More complicated flow control

Print out the information on whether the variable `x` is positive, negative or equals to 0.

How to do it?

Chaining if statements

```
if (x > 0) {  
    System.out.println("x is positive");  
} else if (x < 0) {  
    System.out.println("x is negative");  
} else {  
    System.out.println("x is zero");  
}
```

Chaining is like adding wagons - each is on same level

Nesting if statements

```
if (x == 0) {  
    System.out.println("x is zero");  
} else {  
    if (x > 0) {  
        System.out.println("x is positive");  
    } else {  
        System.out.println("x is negative");  
    }  
}
```



Nesting is like “Matryoshka doll” toy - putting one into another

Chaining vs Nesting

- Both are lexically fine
- Imagine nesting with more options... 4 or 12 or 15...
- And now close all “else” statements
- **Difficult to read and very error prone**

```
if(clientType.equals("newsletter_member")) {  
    price *= 0.9;  
} else if(clientType.equals("regular")) {  
    price *= 0.85;  
} else if(clientType.equals("vip")) {  
    price *= 0.80;  
} else if(clientType.equals("special")) {  
    price *= 0.75;  
} else {  
    price *= 0.95;  
}
```

```
3  
4  
5  
6  
7  
8  
9  
10
```


What about performance?

```
String clientType = "unknown";  
if(clientType.equals("newsletter_member")){  
    price *= 0.9;  
} else if(clientType.equals("regular")){  
    price *= 0.85;  
} else if(clientType.equals("vip")){  
    price *= 0.80;  
} else if(clientType.equals("special")){  
    price *= 0.75;  
} else {  
    price *= 0.95;  
}
```

4 checks before reaching out proper option...

“Switch” is a solution!

```
String clientType = "unknown";
int price = 100;
switch(clientType) {
    case "newsletter_member":
        price *= 0.9;
        break;
    case "regular":
        price *= 0.85;
        break;
    case "vip":
        price *= 0.80;
        break;
    case "special":
        price *= 0.75;
        break;
    default:
        price *= 0.95;
        break;
}
```

Just 1 check before reaching out
propper option....

Note: in “case” it’s possible to use:

- primitives
- Enums
- Strings (from Java7).

Why we use “break” -> try on your
own with different “clientType”

“Switch” is a solution!

```
String clientType = "unknown";
int price = 100;
switch(clientType) {
    case "newsletter_member":
        price *= 0.9;
        break;
    case "regular":
        price *= 0.85;
        break;
    case "vip":
        price *= 0.80;
        break;
    case "special":
        price *= 0.75;
        break;
    default:
        price *= 0.95;
        break;
}
```

- “break” stops block execution and make program to continue after its end
- very useful in switch and all loops

More on flow control...

If you want to see program flow in diagrams visit:

- <https://www.geeksforgeeks.org/decision-making-javaif-else-switch-break-continue-jump/>
- <https://www.geeksforgeeks.org/switch-statement-in-java/>
- <https://www.geeksforgeeks.org/string-in-switch-case-in-java/>

The point of programming is to automate things

Code starts to be useful the moment it's less work to write the code than to do the thing the code is supposed to do by hand.

“Add all numbers from 1 to 100.”

The point of programming is to automate things

Code starts to be useful the moment it's less work to write the code than to do the thing the code is supposed to do by hand.

“Add all numbers from 1 to 100.”

```
public class AddingNumbers{  
  
    public static void main(String args[])  
  
    {  
  
        int sum=0;  
  
        for (int x = 1; x <= 100; x++)  
  
            {sum=sum+x;}  
  
        System.out.println(sum);  
  
    } }  
}
```

The point of programming is to automate things

Code starts to be useful the moment it's less work to write the code than to do the thing the code is supposed to do by hand.

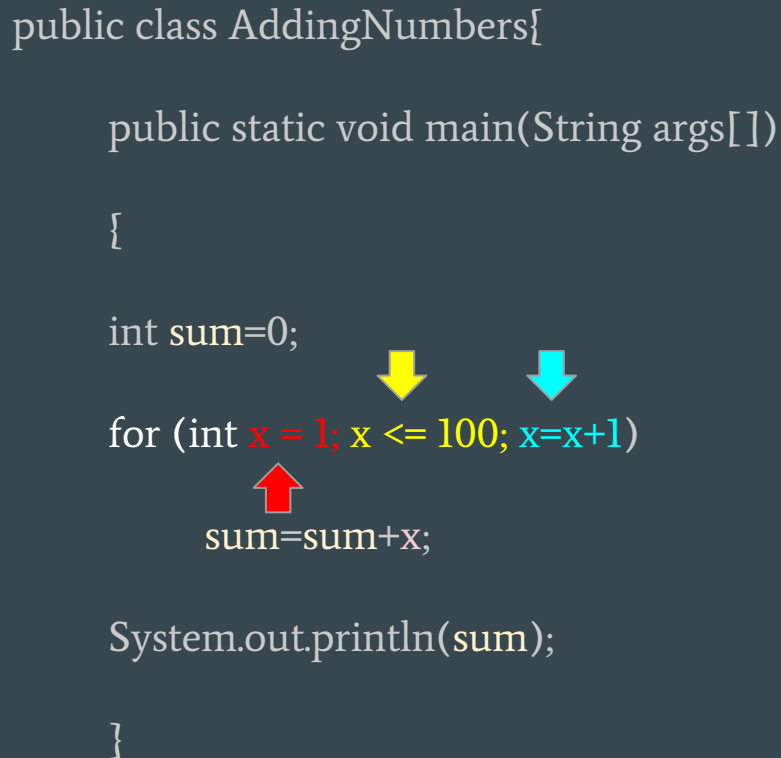
“Add all numbers from 1 to 100.”

FOR ALL X FROM 1 TO 100

INCREMENTING BY ONE

ADD THEM TOGETHER.

```
public class AddingNumbers{  
  
    public static void main(String args[])  
  
    {  
  
        int sum=0;  
  
        for (int x = 1; x <= 100; x=x+1)  
            sum=sum+x;  
  
        System.out.println(sum);  
  
    }  
}
```

A diagram illustrating the execution flow of the provided Java code. A yellow arrow points from the initialization of 'sum' to the start of the 'for' loop. A cyan arrow points from the end of the 'for' loop to the 'System.out.println' statement. A red arrow points from the 'sum=sum+x;' line back to the 'x=x+1;' part of the 'for' loop, indicating the loop's continuation.

The point of programming is to automate things

Code starts to be useful the moment it's less work to write the code than to do the thing the code is supposed to do by hand.

“Add all numbers from 1 to 100.”

FOR ALL X FROM 1 TO 100

INCREMENTING BY ONE

ADD THEM TOGETHER.

```
public class AddingNumbers{  
  
    public static void main(String args[])  
  
    {  
  
        int sum=0;  
  
        for (int x = 1; x <= 100; x=x+1)  
            sum=sum+x;  
  
        System.out.println(sum);  
  
    }  
}
```

how to get from one step to the next

Exercises

Figure out how to write code that:

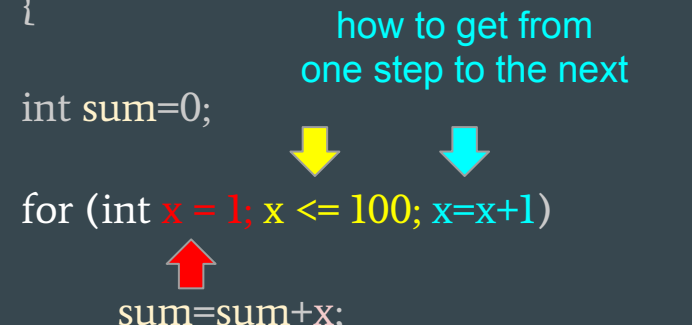
- a) Adds all numbers from 2 to 200
- b) Adds all even numbers from 2 to 200
- c) `int n = 1 + (int)(Math.random() * 10);`

will generate a random integer n from 1 to 10.

Write code that will add all numbers from 0 to 200 that are divisible by n.

```
public class AddingNumbers{  
  
    public static void main(String args[])  
  
    {  
  
        int sum=0;  
  
        for (int x = 1; x <= 100; x=x+1)  
            sum=sum+x;  
  
        System.out.println(sum);  
  
    }  
}
```

how to get from
one step to the next



Arrays

We want the code to automate repetition, and we also want it to store large amounts of data.

How to list items of the same type? For example, we can create a shopping list.

```
String[] ShoppingList={"eggs","bread","milk"};
```

```
for (String x: ShoppingList){  
    System.out.println(x);  
}
```

Arrays

We want the code to automate repetition, and we also want it to store large amounts of data.

How to list items of the same type? For example, we can create a shopping list.

```
String[] ShoppingList={"eggs","bread","milk"};
```

 we have to declare the data type of the elements - the same one for the whole string

```
for (String x: ShoppingList){
```

 we can iterate our "for" loop over elements in the string

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

```
}
```

Arrays

We want the code to automate repetition, and we also want it to store large amounts of data.

How to list items of the same type? For example, we can create a shopping list.

```
String[] ShoppingList={"eggs","bread","milk"};
```

 we have to declare the data type of the elements - the same one for the whole string

```
System.out.println(ShoppingList.length);
```

Arrays

We want the code to automate repetition, and we also want it to store large amounts of data.

How to list items of the same type? For example, we can create a shopping list.

```
String[] ShoppingList={"eggs","bread","milk"};
```

 we have to declare the data type of the elements - the same one for the whole string

```
for (int i=0, i<ShoppingList.length, i++){  
    System.out.println(ShoppingList[i]);  
}
```

Arrays - indexing

```
String[] ShoppingList={"eggs","bread","milk"};
for (int i=0, i<ShoppingList.length, i++){
    System.out.println(ShoppingList[i]);
}
```

How index "i" behaves?

int i	0	1	2
shopping	eggs	bread	milk

Arrays are 0 based and this is a general programming rule.

Arrays have length property - which is equal to number of elements. How many of them we have here?

In Java, the **data type** and the **length** of the array can't be changed. But we can change the **value of the elements**.

```
public class FunWithArrays{  
  
    public static void main(String args[])  
  
    {  
  
        int[] myArray={0,0,0,0,0};  
  
        for (int x: myArray){  
  
            x=(int)(Math.random() * 2);  
  
            System.out.println(x);}  
  
        }  
  
    }
```

In Java, the data type and the length of the array can't be changed. But we can change the value of the elements.

```
public class FunWithArrays{  
  
    public static void main(String args[])  
  
    {  
  
        int[] myArray=new int[5];  
  
        for (int x: myArray){  
  
            x=(int)(Math.random() * 2);  
  
            System.out.println(x);}  
  
        }  
  
    }
```


In Java, the data type and the length of the array can't be changed. But we can change the value of the elements.

```
public class FunWithArrays{
```

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

```
{
```

```
    int[] myArray=new int[5];
```

create an array of length 5 full of 0s

```
    for (int x: myArray){
```

then take each entry in that array

```
        x=(int)(Math.random() * 2);
```

replace it with 0 or 1 at random

```
        System.out.println(x);}

}
```

and print the resulting entry

```
}
```

In Java, the data type and the length of the array can't be changed. But we can change the value of the elements.

```
public class FunWithArrays{
```

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

```
    {
```

```
        int[] myArray=new int[5];
```

create an array of length 5 full of 0s

```
        for (int x: myArray){
```

then take each entry in that array:

```
            x=(int)(Math.random() * 2);
```

{replace it with 0 or 1 at random

```
            System.out.println(x);} 
```

and print the resulting entry}

```
        }
```

```
    }
```

So what if we suddenly need to add an entry to an existing array, but cannot change its length?

Suppose we have an array of 5 integer entries, say

```
int[] myArray={7,23,314,1,97}
```

And want to append 17 at the end.

So what if we suddenly need to add an entry to an existing array, but cannot change its length?

Suppose we have an array of 5 integer entries, say

```
int[] myArray={7,23,314,1,97}
```

And want to append 17 at the end.

```
int[] myArray={7,23,314,1,97}
```

```
int[] newArray=int[myArray.length+1]
```

...

```
newArray[myArray.length]=17
```

So what if we suddenly need to add an entry to an existing array, but cannot change its length?

Suppose we have an array of 5 integer entries, say

```
int[] myArray={7,23,314,1,97}
```

And want to append 17 at the end.

```
int[] myArray={7,23,314,1,97}
```

```
int[] newArray=int[myArray.length+1]
```

...

```
newArray[myArray.length]=17
```

WRITE THAT CODE!

Suppose we have an array of 5 integer entries, say {7,23,314,1,97}, And want to append 17

```
public class FunWithArrays{
    public static void main(String args[])
    {
        int[] myArray = {7,23,314,1,97};
        int oldLength = myArray.length;
        int[] newArray = new int[oldLength+1];

        for (int x=0, x<oldLength,x=x+1){
            newArray[x]=myArray[x];
        }
        newArray[oldLength]=17;
    }
}
```

“While” loops

A while loop repeats as long as a Boolean condition is true. When it is false, the loop stops and the rest of the code is executed.

```
while (1<2) {  
    System.out.println("Doing the loop.");  
}
```

Will print “Doing the loop” forever, because “1<2” never stops being true.

```
while (2<1) {  
    System.out.println("Doing the loop.");  
}
```

Will do nothing.

“While” loops

A while loop repeats as long as a Boolean condition is true. When it is false, the loop stops and the rest of the code is executed.

```
int x=0;
while (x<10) {
    System.out.println("Doing the loop.");
    x=x+1;
}
```


“While” loops

A while loop repeats as long as a Boolean condition is true. When it is false, the loop stops and the rest of the code is executed.

```
int x=0;
while (x<10) {
    System.out.println("Doing the loop.");
    x=x+1;
}
```

Will print “Doing the loop.” 10 times.

`(int)(Math.random()*2)`; comes up with either 0 or 1 at random. It simulates flipping a coin. Can you come up with a while loop that will simulate flipping a coin until “tails”/”1” comes up twice in a row?

`(int)(Math.random()*2)`; comes up with either 0 or 1 at random. It simulates flipping a coin. Can you come up with a while loop that will simulate flipping a coin until “tails”/”1” comes up twice in a row?

There are a lot of possible solutions!

Here's one:

Try to read it line by line

(translate into words!)

to figure out what's going on here.

y is the coin flip. What is x???

```
int x=0;
while (x<2) {
    int y=(int) (Math.random()*2);
    System.out.println(y);
    x=x+y;
    if (x==1 && y==0) {
        x=0;
    }
}
```

More on loops...

If you want to see program flow in diagrams visit:

- <https://www.geeksforgeeks.org/loops-in-java/>
- <https://www.geeksforgeeks.org/for-each-loop-in-java/>
- <https://www.geeksforgeeks.org/loop-java-important-points/>

Exercises

Online editor:

Examples: <http://tpcg.io/swjfMA>

Exercises: <http://tpcg.io/Y3xmfl>

Github:

<https://github.com/bjowczarek/workshop-lesson-3>

EXERCISES

Exercise 1 <http://tpcg.io/oKkofA>

Exercise 2 <http://tpcg.io/KGcx6Q> // requires switch [some help on switch here](#)

Exercise 3 <http://tpcg.io/io9IEr> // solutions at the end of the file

Exercise 4 <http://tpcg.io/3JGbg2> // solutions at the end of the file, [help on Strings](#)

Those exercises are also on Bartek's github

<https://github.com/bjowczarek/workshop-lesson-3/>