Introduction to Information Systems and Programming

Java Introduction

Objectives

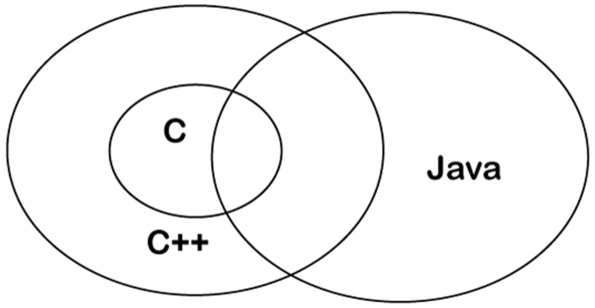
- Java Intro
- Variables
- BigDecimal
- Array
- String
- If-then
- For loop
- For-each loop
- While loop
- Method

Why Java?

- Stable - This programming language has been around since 1995. It has stood the test of time.
- Versatile – Has been used for a wide range of applications such as web development, enterprise systems, cloud computing, scientific research, desktop application, and mobile apps (Android).
- Large community support – can easily find resources such as libraries, tutorial, forum QnA, etc

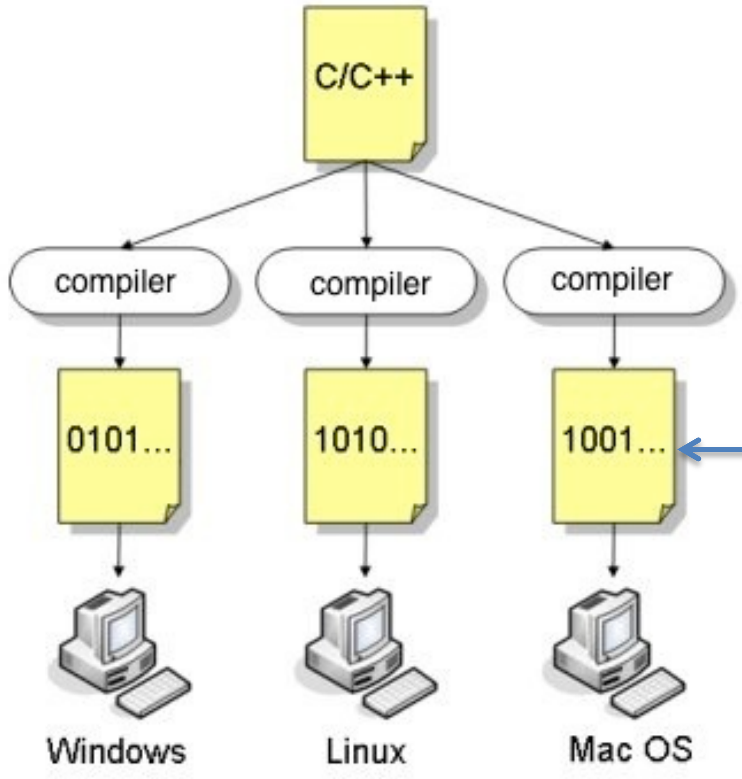
Why Java?

Statically-typed language – Unlike Python which is dynamically-typed language, many other popular languages such as Java, C, C#, C++, Swift, Go, Kotlin, etc. are statically-typed language. After learning Java, you can learn many other programming language faster now.

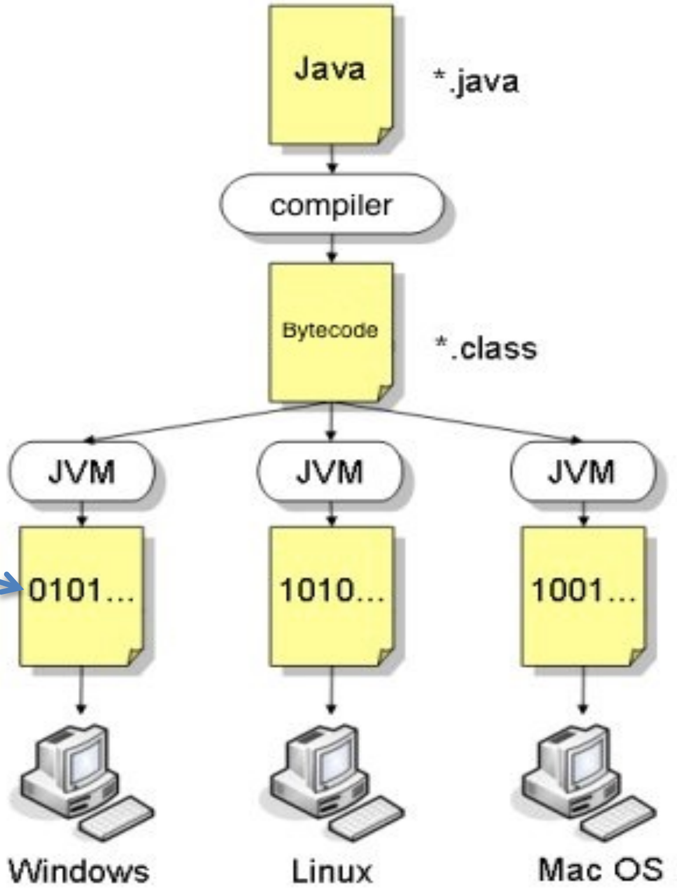


How Java Code is Executed

High-level programming languages have to be translated into machine-language before the program can be executed by compiler



Machine codes are platform-specific, not portable, not suitable for connected computers

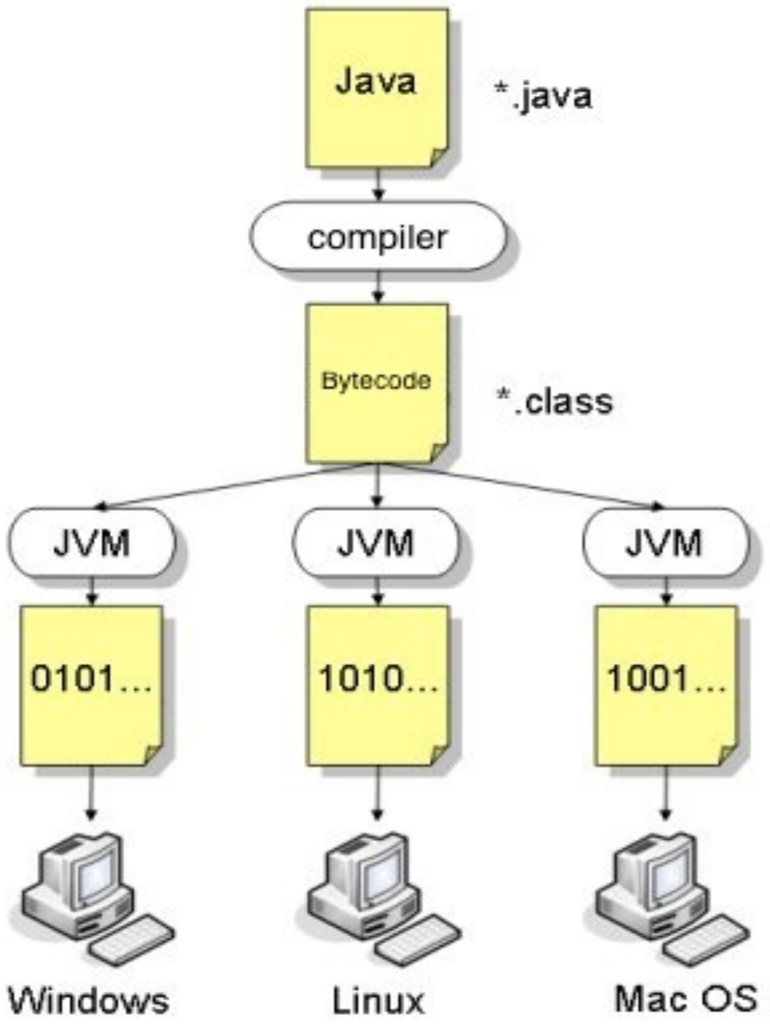


About JAVA’s portability

- Java programs are compiled into a specific type of machine language called **bytecode**
- Running *javac JavaFile.java* in terminal/cmd will create the JavaFile.class file, which is the bytecode file.

- Bytecode can be run on any computer with a JVM
- JVM: a software that interprets Java bytecode
- Running *java JavaFile* will execute the file

- JVM is installed when you installed JDK.
- JDK is a development environment to compile and run a program developed using the Java programming language. It is similar to how you install Anaconda for Python programming.



Variables

- Variables are used to store data in a program
- There are 4 types of variables:
  - Non static: instance variable
  - Static: class variable
  - Local: exists only in a between square bracket { }
  - Parameters: the input variable of a method.

  (Read more here: https://dev.java/learn/language-basics/variables/)

Variables

- In Java, you have to specify what type of data a variable represents. That is why it is called **Statically-typed**
- For example

```
int a = 1;
```

- The code above means that a variable named a of type integer is declared and assigned with value 1

Variables

```
public class VariablesDemo {
    public static void main(String[] args) {
        int a = 1;
        boolean b = true;
        double c = 4.3;
        float d = 2.9f;
        char e = 'e';

        System.out.print(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(d);
        System.out.println(e);
    }
}
```

public -> 1 of 4 access modifiers in java (public, protected, default, and private). public allows class, attributes, or method to be accessible from anywhere (even from different package).

class -> syntax to define a class

static -> set attribute (or method) to class attributes instead of instance/object attributes

void -> Method output datatype is “None”

main -> Main function of the Java Class. When you execute a java file, JVM is looking for the main function of the class whose name is the same as the file name

String[] -> Array of String

args -> parameter name
; -> semicolon ends a statement

System.*out*.print -> equivalent to print(, end=’’) in python
System.*out*.println -> equivalent to print() in python

Variables

```
public class VariablesDemo {
    public static void main(String[] args) {
        int a = 1;
        boolean b = true;
        double c = 4.3;
        float d = 2.9f;
        char e = 'e';

        System.out.print(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(d);
        System.out.println(e);
    }
}
```

public -> 1 of 4 access modifiers in java (public, protected, default, and private). public allows class, attributes, or method to be accessible from anywhere (even from different package).

class -> syntax to define a class

static -> set attribute (or method) to class attributes instead of instance/object attributes

void -> Method output datatype is "None"

main -> Main function of the Java Class. When you execute a java file, JVM is looking for the main function of the class whose name is the same as the file name

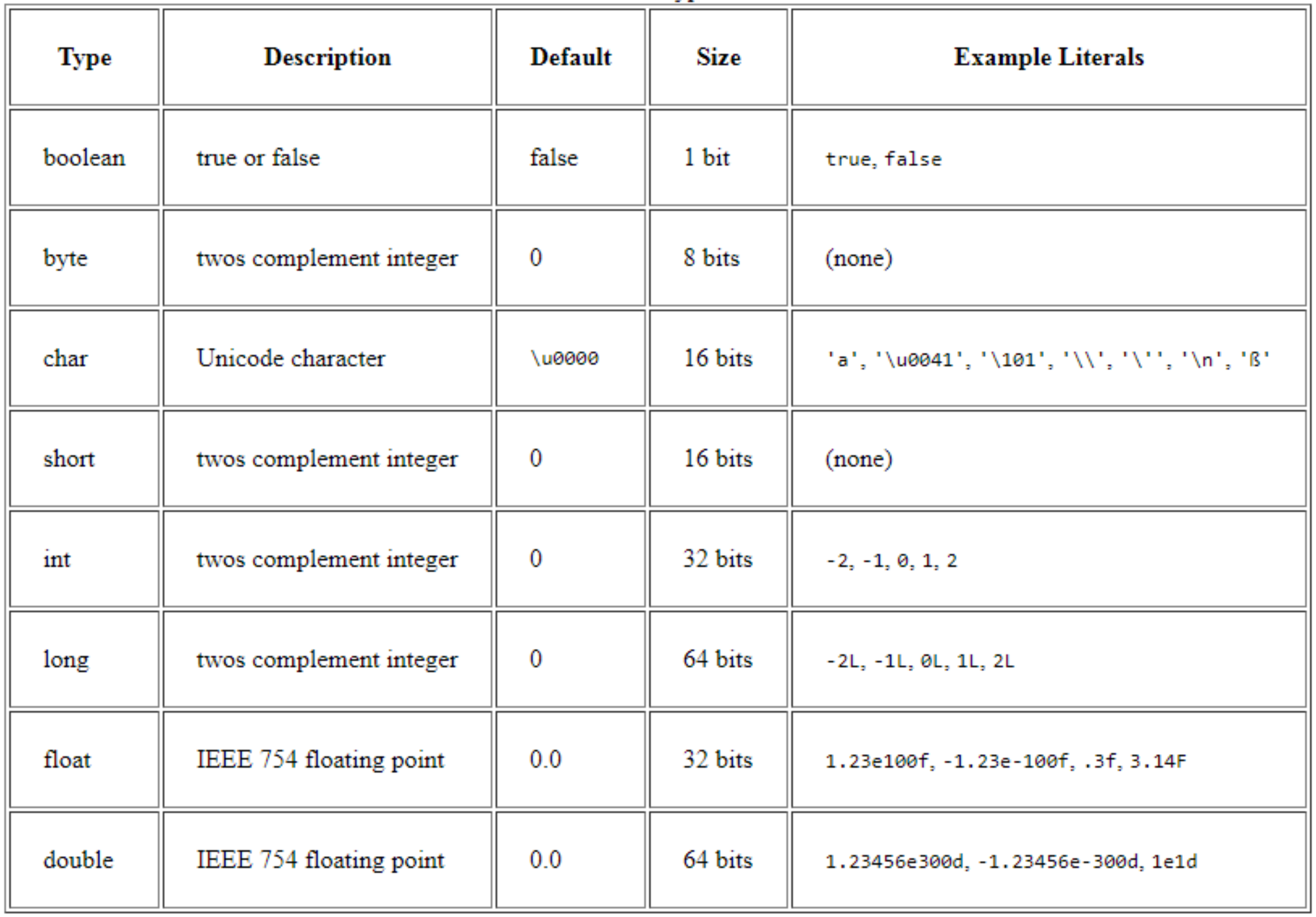
String[] -> Array of String

args -> parameter name
; -> semicolon ends a statement

System.*out*.print -> equivalent to print(, end='') in python
System.*out*.println -> equivalent to print() in python

Android Studio Shortcut:
public static void main(String[] args) can be typed using a shortcut -> type "main", and then press enter when autocomplete show up.
System.out.println(); also has a shortcut: type "sout" then press enter.

Primitive data types in Java



Type	Description	Default	Size	Example Literals
boolean	true or false	false	1 bit	`true`, `false`
byte	twos complement integer	0	8 bits	(none)
char	Unicode character	`\u0000`	16 bits	`'a'`, `'\u0041'`, `'\101'`, `'\\'`, `'\''`, `'\n'`, `'ß'`
short	twos complement integer	0	16 bits	(none)
int	twos complement integer	0	32 bits	`-2`, `-1`, `0`, `1`, `2`
long	twos complement integer	0	64 bits	`-2L`, `-1L`, `0L`, `1L`, `2L`
float	IEEE 754 floating point	0.0	32 bits	`1.23e100f`, `-1.23e-100f`, `.3f`, `3.14F`
double	IEEE 754 floating point	0.0	64 bits	`1.23456e300d`, `-1.23456e-300d`, `1e1d`

BigDecimal Object

- Using primitive data types might result in rounding error. Try:

```
double a = 0.05;
double b = 0.06;
double c = b-a;
```

- BigDecimal is an object with Arbitrary-precision
- It can represent exact value of a number
- Suitable in the context of finance
- Try the following:

```
BigDecimal X = new BigDecimal("0.05");
BigDecimal Y = new BigDecimal("0.06");
BigDecimal Z = new BigDecimal(0.05);
// There are various ways to instantiate BigDecimal object

System.out.println( Y.subtract(X) );
```

array

- Array is used to store a **collection** of **same-type** data
- Array is an object, not primitive datatype
- Index starts from 0

Read more on:
https://dev.java/learn/language-basics/arrays/#copying

2. array

See code below on how to declare, assign, access, and modify array object

```java
int[] c; // declare a variable to reference an array
c = new int[3];  // allocate memory for 3 int
c[0]=17; c[1]=23; c[2]=38; // assign value.

// The three lines above can be written in one line like this
// int[] c = {17, 23, 38};

System.out.println("0th value: " + c[0]);
System.out.println("1st value: " + c[1]);
System.out.println("2nd value: " + c[2]);


Output:
0th value: 17
1st value: 23
2nd value: 38
```

Equals vs ==

- Array is not a primitive datatype
- We should use **.equals** to compare the content of two non-primitive objects. Since the objects are arrays in this case, we should use Arrays.equals (import java.util.Arrays; beforehand)
- We can use == to compare **primitive datatype** objects
- == compares the identity hash code of **reference variable**

Equals vs ==

```
int[] c; // declare a variable to reference an array
c = new int[3];  // allocate memory for 3 int
c[0]=17; c[1]=23; c[2]=38; // assign value.

int[] e = {17, 23, 38};
System.out.println( "Equal: " + Arrays.equals(e, c) );
System.out.println( "==: " + (e == c) );


Output:
Equal: true
==: false
```

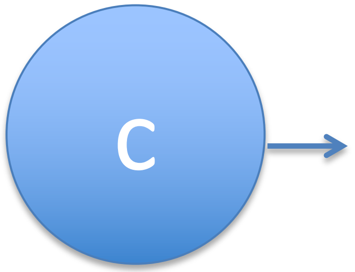
Why == gives you false here?
because **e** and **c** refers to different object in memory

Aliasing

```
int[] c; // declare a variable to reference an array
c = new int[3];  // allocate memory for 3 int
c[0]=17; c[1]=23; c[2]=38; // assign value.

int[] d = c; // aliasing
d[0] = 999;

System.out.println("0th value: " + c[0]);
System.out.println("1st value: " + c[1]);
System.out.println("2nd value: " + c[2]);


Output:
0th value: 999
1st value: 23
2nd value: 38
```

Array – what's happening behind

```
int[] c;
c = new int[3];
c[0]=17;
c[1]=23;
```

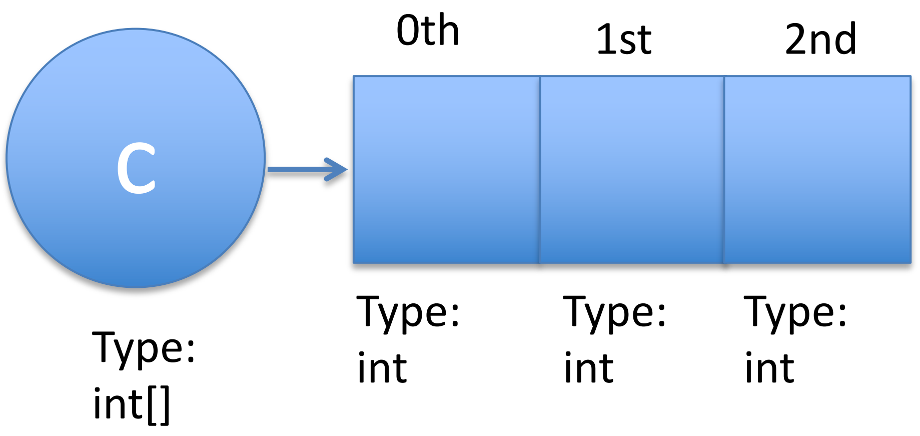


Type:
int[]

c is declared as a variable of reference to an array (of integer)

Array – what's happening behind

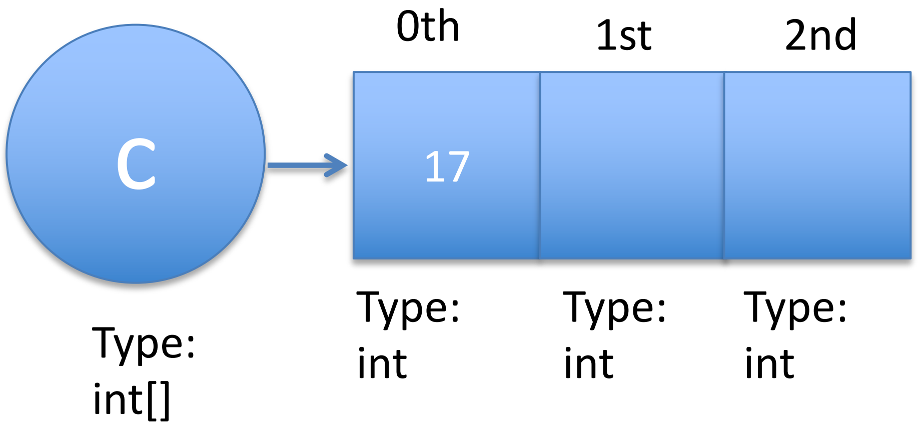
```
int[] c;
c = new int[3];
c[0]=17;
c[1]=23;
```



Three int memory are allocated. The array reference is assigned to c

Array – what's happening behind

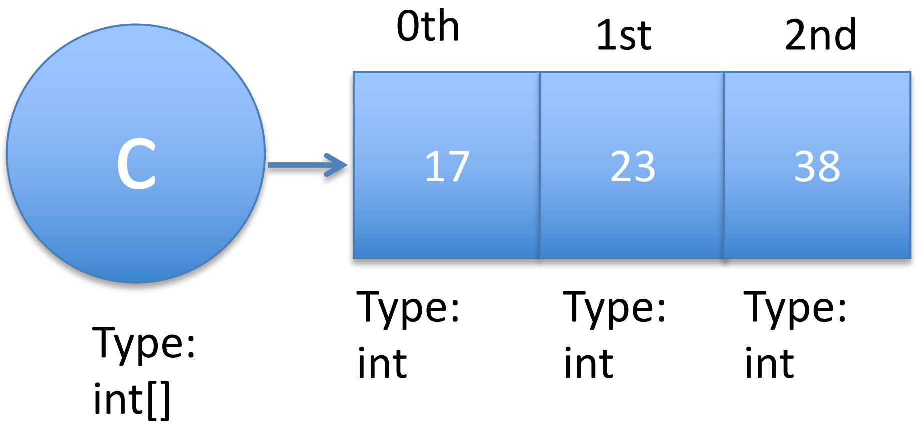
```
int[] c;
c = new int[3];
c[0]=17;
c[1]=23;
```



We assign value 17 to the c[0] memory cell
(not assign 17 to c. c is just a reference)

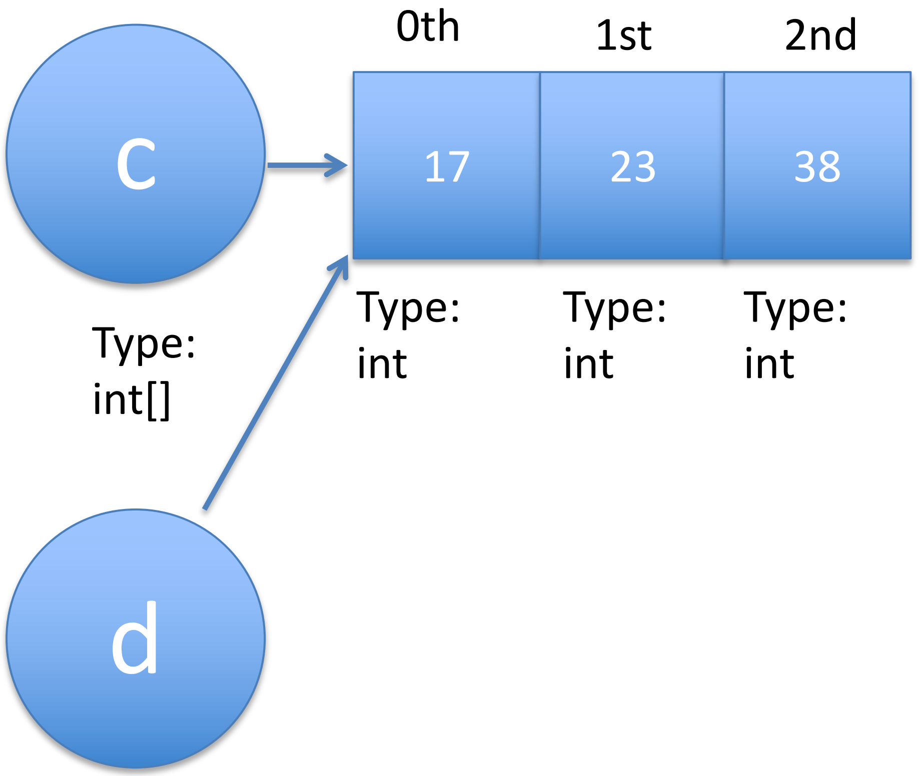
Array – what's happening behind

```
int[] c;
c = new int[3];
c[0]=17;
c[1]=23;
c[2]=38;
```



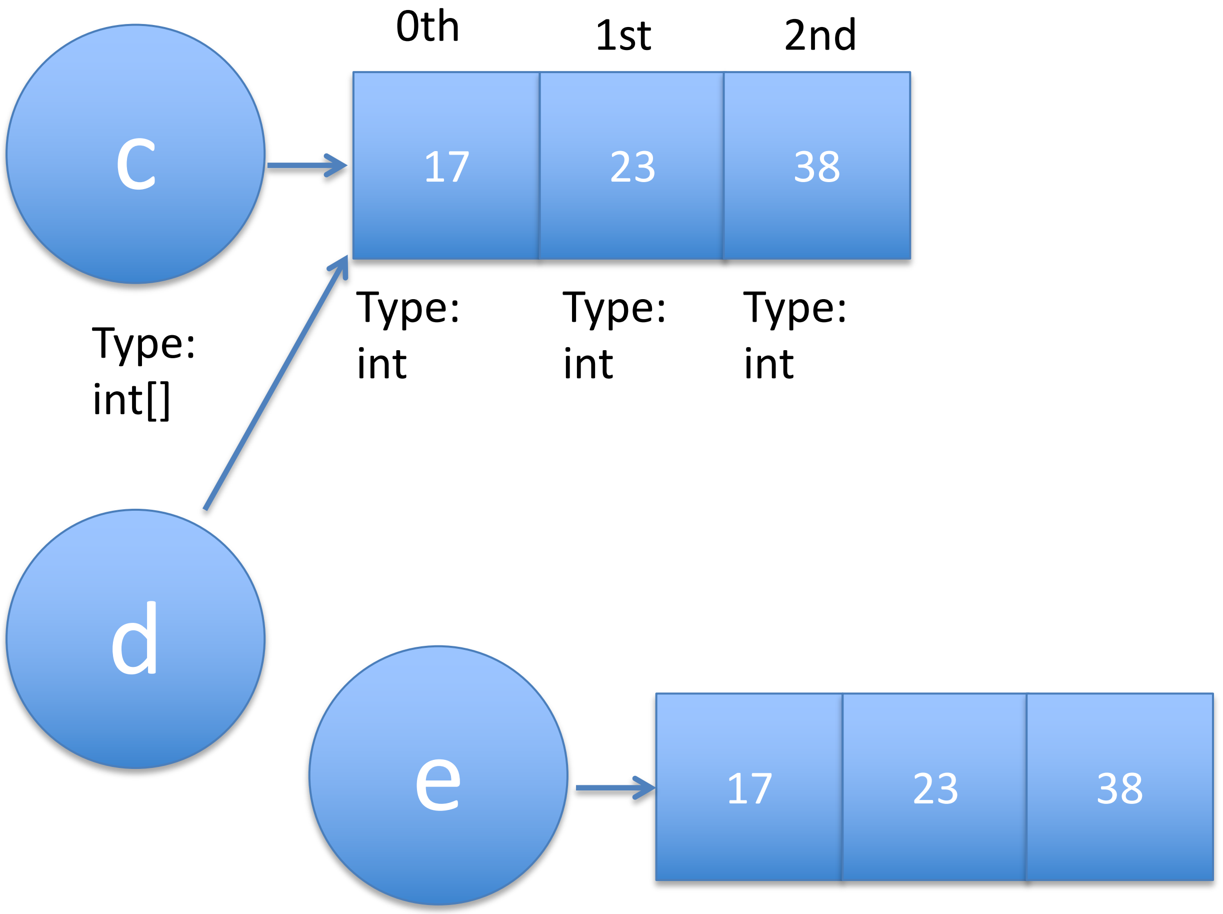
Array – what's happening behind

```
int[] c;
c = new int[3];
c[0]=17;
c[1]=23;
c[2]=38;

int[] d = c;
```



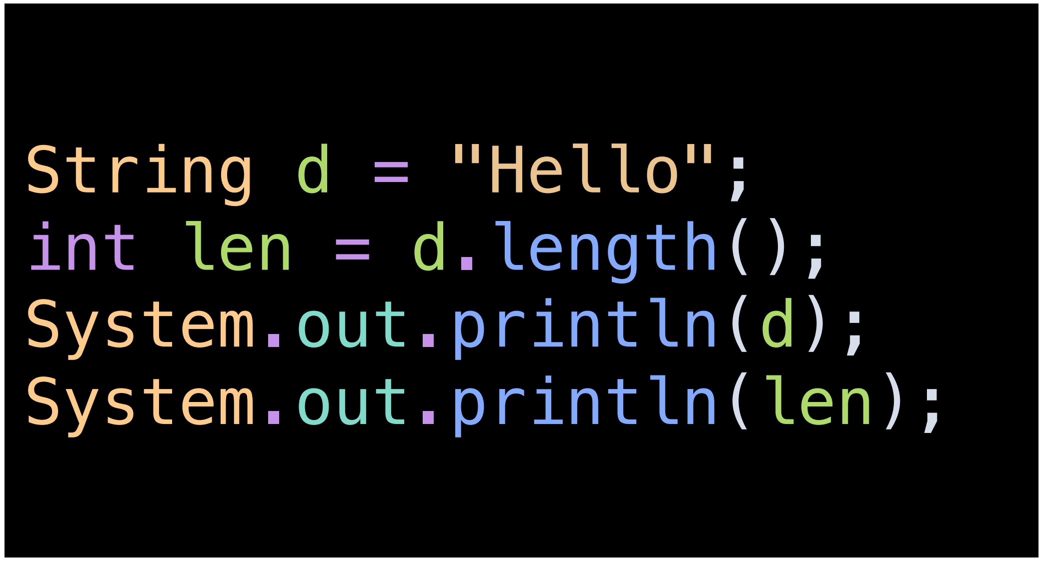
Array – what's happening behind

```
int[] c;
c = new int[3];
c[0]=17;
c[1]=23;
c[2]=38;

int[] d = c;
int[] e = {17, 23, 38};
```



String

- String is a sequence of characters.
- Immutable



```
String d = "Hello";
int len = d.length();
System.out.println(d);
System.out.println(len);
```

String

Alternative way to create a String object

```
char[] dArray = { 'h', 'e', 'l', 'l', 'o' };
String dString = new String(dArray);
System.out.println(dString);
```

If then

If-then-else: selective statements and conditional execution

```java
int simonWeight = 200;  // in kg. OH NO….
String advice = "";       // this is an empty string
if (simonWeight <= 75) {
    advice = "fit";
} else if (simonWeight <= 100) {
    advice = "eat less";
} else if (simonWeight <= 150) {
    advice = "no dinner";
} else {
    advice = "no dinner no breakfast no tea no lunch";
}
System.out.println(advice);
```

What is the advice?

for loop

Loop: program construct to control repeated execution of a block of statements

Initial action

Loop continuation condition

Action after each iteration

```
for(int i=1; i<=4; i++){
    System.out.println("Count is: " + i);
}
```

for loop

Loop: program construct to control repeated execution of a block of statements

Initial action

Loop continuation condition

Action after each iteration

```
for(int i=1; i<=4; i++){
    System.out.println("Count is: " + i);
}
```

The output will be:

```
Count is: 1
Count is: 2
Count is: 3
Count is: 4
```

for-each loop

Loop through a collection of items

```java
public static void main(String[] args) {
    double[] numbers = {1.5, 3.0, 4.5};

    for (double number: numbers) {
        System.out.println(number);
    }
}
```

Output:

```
1.5
3.0
4.5
```

while loop

Loop through a collection of items

```
public static void main(String[] args) {
    int x = 0;

    while (x <= 5)
    {
        System.out.println("x = " + x);
        x += 1;
    }
}
```

Output:

```
x = 0
x = 1
x = 2
x = 3
x = 4
x = 5
```

Method

- **method** is a function inside a class
- **method signature** is method name + its parameters (in the code below, `printNTimes(int n, String message)` is the signature)
- Declaring parameter is the same as declaring variable (type is needed)
- Cannot set default values for parameters in java
- The printNTimes method is invoked in the main function by passing 5 and “HOHO” as the arguments.

```java
public class MethodDemo {
    public static void main(String[] args) {
        printNTimes(5, "HOHO");
    }

    public static void printNTimes(int n, String message) {
        for (int i=0; i<n; i++) {
            System.out.println(message);
        }
    }
}
```

Method

Discussion:

Parameter vs Argument?

What happen if static modifier is removed for the printNTimes method definition?

What needs to be done to invoke printNTimes if the static modifier was removed?

What happen if we uncomment the **return message;** line.

```java
public class MethodDemo {
    public static void main(String[] args) {
        printNTimes(5, "HOHO");
    }

    public static void printNTimes(int n, String message) {
        for (int i=0; i<n; i++) {
            System.out.println(message);
        }
//          return message;
    }
}
```

Reference of JAVA syntax

Some materials for reference

- https://dev.java/learn/language-basics/
- http://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html