INTRODUCTION TO JAVA PROGRAMMING

Summary

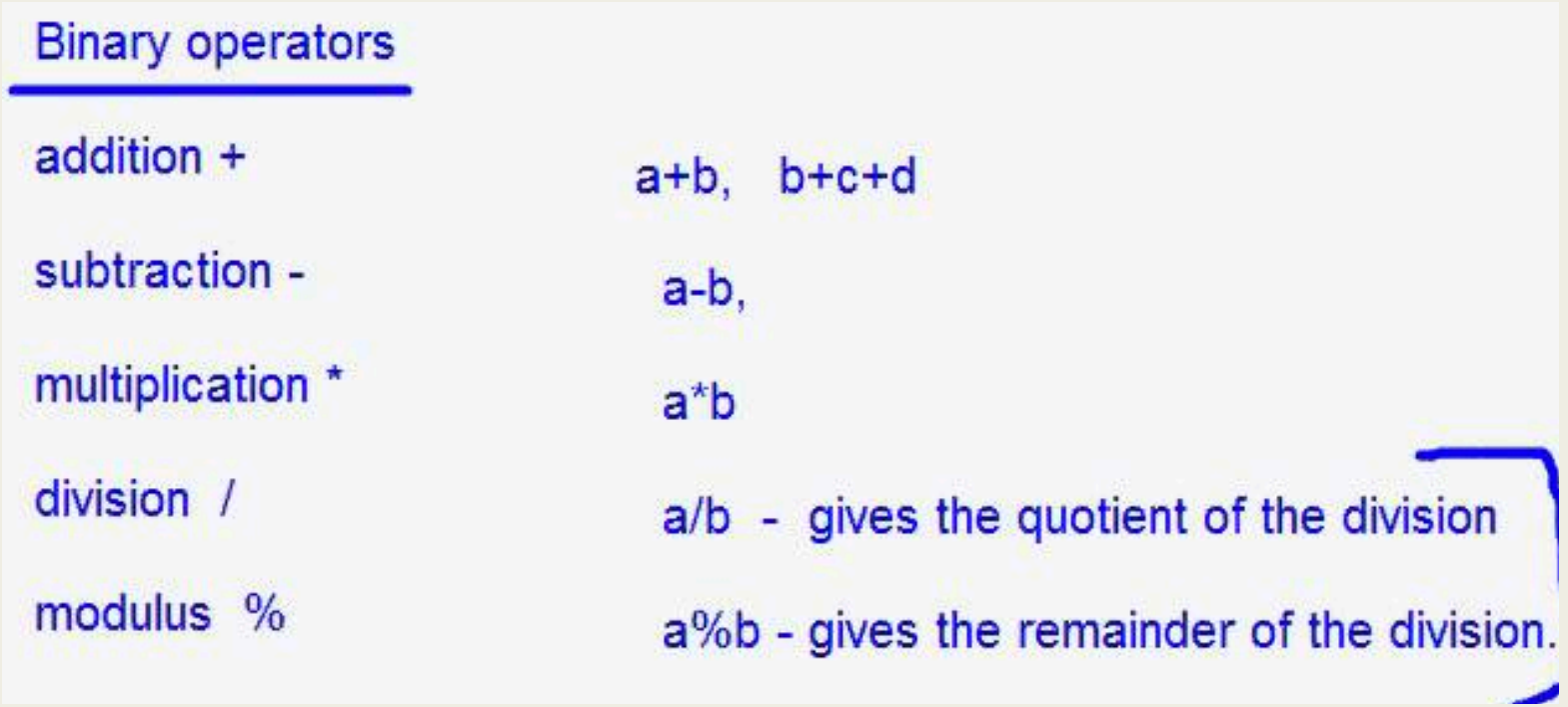
- You will learn about Java operators/expressions/statements
- You will be familiar with data types and Strings in Java
- You will know to operate with Date and Calendar and use conversion
- We will recap conditionals, loops, methods and arrays

Basic notions

- **Operators**
- **Expressions**
- **Statements**

Operators

- **Operators** are special symbols that are used to represent simple computations like *addition* and *multiplication*.
- Most of the operators in Java do exactly what you would expect them to do, because they are common mathematical symbols.



Operators

- **Operators** are special symbols that are used to represent simple computations like addition and multiplication.
- Most of the operators in Java do exactly what you would expect them to do, because they are common mathematical symbols.

```
int hour, minute;
hour = 11;
minute = 59;
System.out.print ("Number of minutes since midnight: ");
System.out.println (hour*60 + minute);
System.out.print ("Fraction of the hour that has passed: ");
System.out.println (minute/60);
```

Order of operations

- When more than one operator appears in an expression the order of evaluation depends on the rules of **precedence.** A complete explanation of precedence can get complicated, but just to get you started:
  - *Multiplication* and *division* take precedence (happen before) *addition* and *subtraction*.
- If the operators have the same precedence they are evaluated from left to right.
- Any time you want to override the rules of precedence (or you are not sure what they are) you can use **parentheses.**

Expression

- A combination of variables, operators and values that represents a single result value. Expressions also have types, as determined by their operators and operands.

```
if( expression ) {
  statement 1;
  statement 2;
}
```

```
if( a + b == c ) {
  statement 1;
  statement 2;
}
```

```
d = a + b == c
if( d ) {
  statement 1;
  statement 2;
}
```

Statement

- A line of code that represents a command or action.
- So far, the statements we have seen are declarations, assignments, and print statements.

Data types

- **Primitive**
- **Wrappers**
- **Strings**
- **Date**
- **Calendar**
- Math lib

Data types

- Java has a variety of datatypes
  - *short, int, long, byte, float, double, char, boolean,..*
- It has multiple wrapper classes
  - *Short, Integer, Long, Byte, Float, Double, Character, Boolean,..*
- Other class based types
  - *String, Date*
  - *Your own type e.g. Person, Car, Money*

  - *int vs Integrer!*
- plain type vs object type (wrapper)
- Integer can be null, but occupies my space
- Integration with Java Collections (Set<Integer>, List<String>)

int vs Integer

- **int** is a primitive type.
- Variables of type **int** store the actual binary value for the **integer** you want to represent.
- **int**.parseInt("1") doesn't make sense because **int** is not a class and therefore doesn't have any methods.
- **Integer** is a class, no different from any other in the Java language.
- Integer can be null, but occupies more space

Wrapper types

java.lang.Integer
-value: int +MAX_VALUE: int +MIN_VALUE: int
+Integer(value: int) +Integer(s: String) +byteValue(): byte +shortValue(): short +intValue(): int +longVlaue(): long +floatValue(): float +doubleValue():double +compareTo(o: Integer): int +toString(): String +valueOf(s: String): Integer +valueOf(s: String, radix: int): Integer +parseInt(s: String): int +parseInt(s: String, radix: int): int

java.lang.Double
-value: double +MAX_VALUE: double +MIN_VALUE: double
+Double(value: double) +Double(s: String) +byteValue(): byte +shortValue(): short +intValue(): int +longVlaue(): long +floatValue(): float +doubleValue():double +compareTo(o: Double): int +toString(): String +valueOf(s: String): Double +valueOf(s: String, radix: int): Double +parseDouble(s: String): double +parseDouble(s: String, radix: int): double

Wrapper types

- Constructors
  - *public Integer(int value)*
  - *public Integer(String s)*
- Constants
  - *MAX_VALUE and MIN_VALUE*
- Conversions
  - *doubleValue()*
  - *floatValue()*
  - *intValue()*
  - *Double doubleValue = Double.valueOf(“13.7”)*

Chars and Strings

```
char fred = 'c';
if (fred == 'c') {
  System.out.println (fred);
}
```

```
String fruit = "banana";
char letter = fruit.charAt(1);
System.out.println (letter);
int length = fruit.length();
```

Strings

```
String f1 = "banana";
String f2 = ”coco";
System.out.println(f1.equals(f2));
```

```
String str = "abc";
//is equivalent to:
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

```
System.out.println("abc" + cde);
String c = "abc".substring(2,3);
String d = cde.substring(1, 2);
```

https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

Strings API (subset)

```
charAt(int index);
compareTo(String s);
concat(String s);
contains(CharSequence s);
endsWith(String suffix);
format(..);
getBytes(Charset encoding);
hashCode();
indexOf(char c);
isEmpty();
lastIndexOf(char c);
length();
replace(CharSequence s1,
                CharSequence s2);
split(String regex);
startWith(String s);
substring(int beg; int end);
toLowerCase();
toUperCase();
valueOf(ling l);
```

https://docs.oracle.com/javase/8/docs/api/java/util/Date.html

Date

- The class Date represents a specific instant in time, with millisecond precision.
- **Date**() Allocates a Date object and initializes it so that it represents the time at which it was allocated, measured to the nearest millisecond.
- **Date**(long date) Allocates a Date object and initializes it to represent the specified number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 GMT.
- Functions
  - *after, before, getTime*
  - *Many deprecated methods -> use class Calendar*

Data formatting

https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

```
public class DateTest {
  public static void main(String[] args) {
    Date now = new Date();
    System.out.println("toString(): " + now);
    SimpleDateFormat df
          = new SimpleDateFormat("E, y-M-d 'at' h:m:s a z");
    System.out.println("Format 1: " + df.format(now));

    df = new SimpleDateFormat("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
    System.out.println("Format 2: " + df.format(now));

    df = new SimpleDateFormat("EEEE, MMMM d, yyyy");
    System.out.println("Format 3: " + df.format(now));
  }
}
```

```
toString(): Sat Sep 25 21:27:01 SGT 2010
Format 1:   Sat, 10-9-25 at 9:27:1 PM SGT
Format 2:   Sat 2010.09.25 at 09:27:01 PM SGT
Format 3:   Saturday, September 25, 2010
```

Data formatting

https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

```
//To parse a text string int Date, use:
DateFormat formatter = ....
Date myDate = formatter.parse(myString);
```

Measure time

```
// Measuring elapsed time
long startTime = System.currentTimeMillis();
// The code being measured
  .......
long estimatedTime = System.currentTimeMillis() - startTime;
```

https://docs.oracle.com/javase/8/docs/api/java/util/Calendar.html

Calendar

- The Calendar class is an abstract class that provides methods for converting between a specific instant in time and a set of calendar fields such as YEAR, MONTH, DAY_OF_MONTH, HOUR, and so on, and for manipulating the calendar fields, such as getting the date of the next week.

Calendar usage

```
// Get the year, month, day, hour, minute, second
import java.util.Calendar;

public class GetYMDHMS {
  public static void main(String[] args) {
    Calendar cal = Calendar.getInstance();
    // You cannot use Date class to extract individual Date fields
    int year = cal.get(Calendar.YEAR);
    int month = cal.get(Calendar.MONTH);       // 0 to 11
    int day = cal.get(Calendar.DAY_OF_MONTH);
    int hour = cal.get(Calendar.HOUR_OF_DAY);
    int minute = cal.get(Calendar.MINUTE);
    int second = cal.get(Calendar.SECOND);
    System.out.printf(
        "Now is %4d/%02d/%02d %02d:%02d:%02d\n",  // Pad with zero
        year, month+1, day, hour, minute, second);
  }
}
```

Calendar usage

```
..
// Manipulate
Calendar calTemp;
calTemp = (Calendar) cal.clone();
calTemp.add(Calendar.DAY_OF_YEAR, -365);
System.out.println("365 days ago, it was: " + calTemp.getTime());
calTemp = (Calendar) cal.clone();
calTemp.add(Calendar.HOUR_OF_DAY, 11);
System.out.println("After 11 hrs, it will be: " + calTemp.getTime());

// Roll
calTemp = (Calendar) cal.clone();
calTemp.roll(Calendar.HOUR_OF_DAY, 11);
System.out.println(
      "Roll 11 hours, it will be: " + calTemp.getTime());
System.out.println();
```

Math library

- Explore Math library and print these
- int x = (int) Math.PI;
- float y = Math.PI;
- double root = Math.sqrt (17.0);
  double angle = 1.5;
  double height = Math.sin (angle);

```
import java.util.Random;
..
Random rand = new Random();
int n = rand.nextInt(50) + 1;
//50 is the maximum and the 1 is our minimum
```

Structuring & control

- **Methods**
- **Conditionals**
- **Loops**
- **Recursion**
- **Array**

Methods

- Logically structuring a class

A method

```
public class Sum {
	public static void main(String[] args) {
		int a = 3;
		System.out.println(a+7);
	}
}
```

Methods

- Logically structuring a class

```
public class Sum {

  private static int initA() {return 3;}

  public static void main(String[] args) {
     int a = initA();
     System.out.println(a+7);
  }
}
```

Methods

- Logically structuring a class

```
public class Sum {

   private static int initA() {return 3;}
   private static void print(int x) {
      System.out.println(x);
   }

   public static void main(String[] args) {
      int a = initA();
      print(a+7);
   }
}
```

Methods

- Anatomy

Method with a return type

Method with a parameter

```
public class Sum {

   private static int initA() {return 3;}
   private static void print(int x) {
      System.out.println(x);
   }

   public static void main(String[] args) {
      int a = initA();
      print(a+7);
   }
}
```

Division / modulo

- There is a difference
  - int quotient = 7 / 3;
  - int remainder = 7 % 3;

- What is the difference?
-

Conditionals

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

```
x == y                  // x equals y
x != y                  // x is not equal to y
x > y                   // x is greater than y
x < y                   // x is less than y
x >= y                  // x is greater than or equal to y
x <= y                  // x is less than or equal to y
```

Loop

1. Imagine a situation like this below when increasing by one till ten

```
int a=1; print(a);
a=a+1; print(a);
a+=1; print(a);
a++; print(a); // post incr
..
++a; print(a); // pre incr
```

Loop

1. Remember we want to be efficient

```
for (int a = 1; a<=10; a++) {
     print(a);
}
```

Loop

1. Remember we want to be efficient but we have options

```
for(int a = 1; a<=10; a++) {
    print(a);
}
```

```
int a = 1
while(a<=10) {
    a++;
    print(a);
}
```

Wrong! Where is a bug?

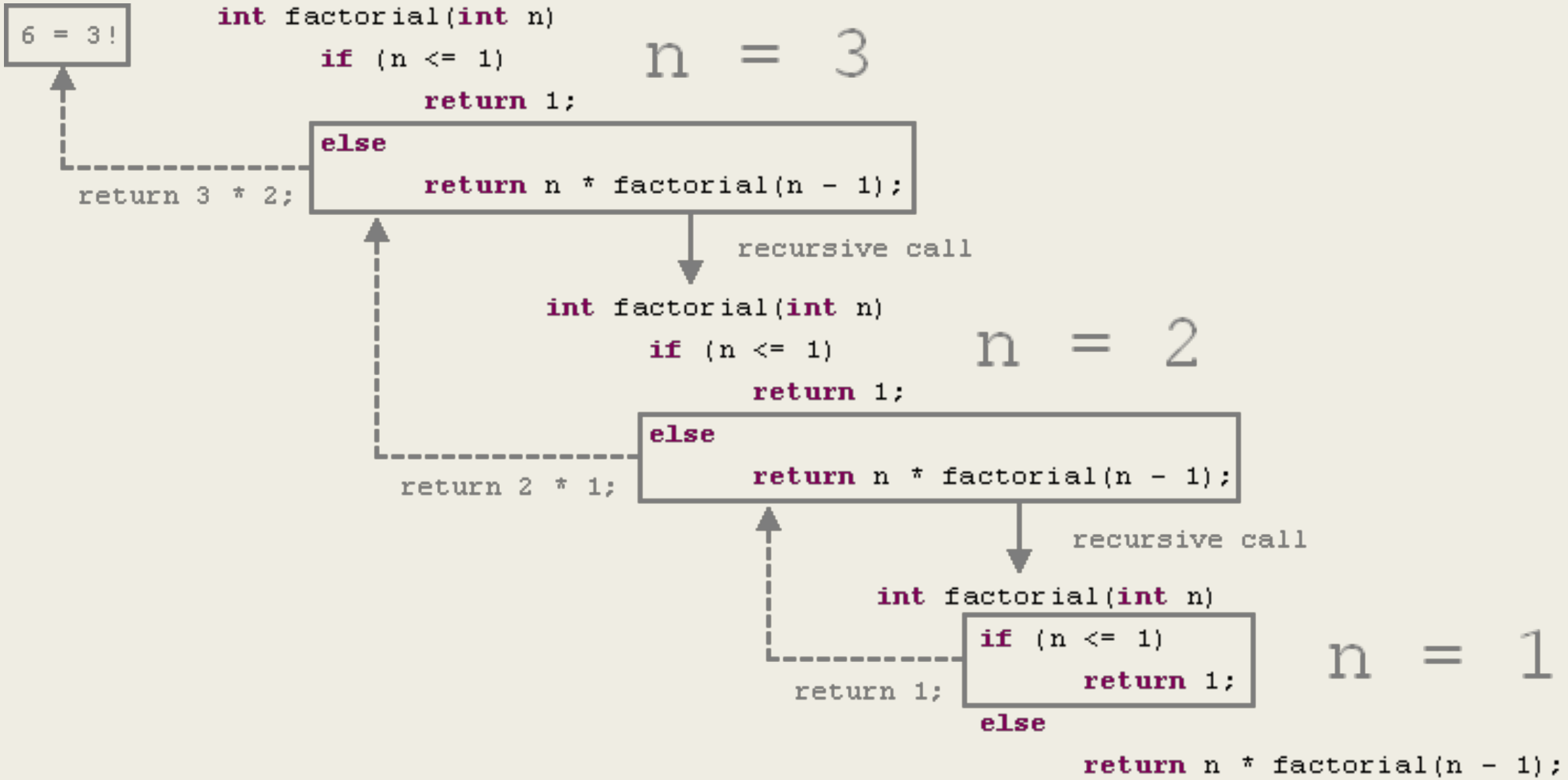
Another option to make a loop, a recursion

1. Repeated method call until condition holds

```
public static void countdown (int n) {
      if (n == 0) {
            System.out.println (”Yes!");
      } else {
            System.out.println (”n=” + n);
            countdown (n-1);
      }
}
```

Practical example : Factorial

```
4! = 4*3*2*1 = 24
5! = 5*4*3*2*1 = 120
```


6 = 3!
int factorial(int n)
if (n <= 1)
return 1;
n = 3
else
return n * factorial(n - 1);
return 3 * 2;
recursive call
int factorial(int n)
if (n <= 1)
return 1;
n = 2
else
return n * factorial(n - 1);
return 2 * 1;
recursive call
int factorial(int n)
if (n <= 1)
return 1;
n = 1
return 1;
else
return n * factorial(n - 1);

Factorial as a recursion

Implement and debug

1. Call until

```
class Factorial {
  static int factorial(int n){
     if (n == 0) return 1;
     else return(n * factorial(n-1));
  }
  public static void main(String args[]){
    int number = 4;//It is the number to calculate factorial
    fact = factorial(number);
    System.out.println(" “+number + ”! is: “ + fact);
  }
}
```

Factorial as a loop

1. Loop until (dynamic programming)

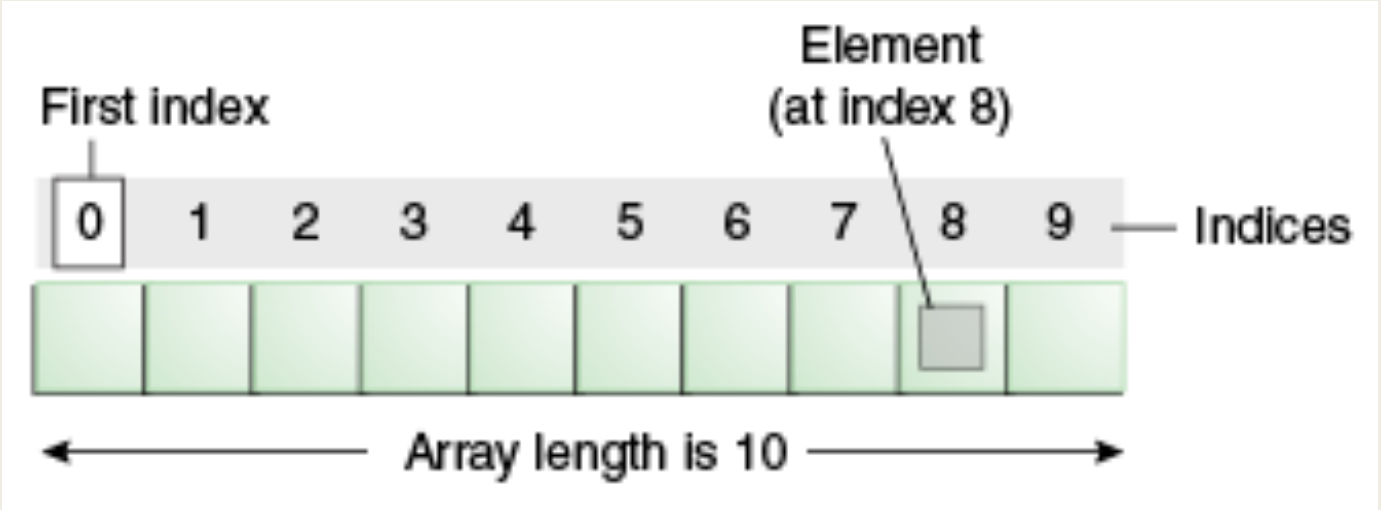
```
class FactorialExample{
     public static void main(String args[]){
          int i, fact = 1;
          int number = 5;//It is the number to calculate factorial
          for (i = 1; i <= number; i++) {
               fact = fact * i;
          }
          System.out.println("" + number + "! is: "+fact);
     }
}
```

Array 0 dimensional (or 1?)

```
int a = 1;
```

Array 1 dim..

```
int a = 1;
```

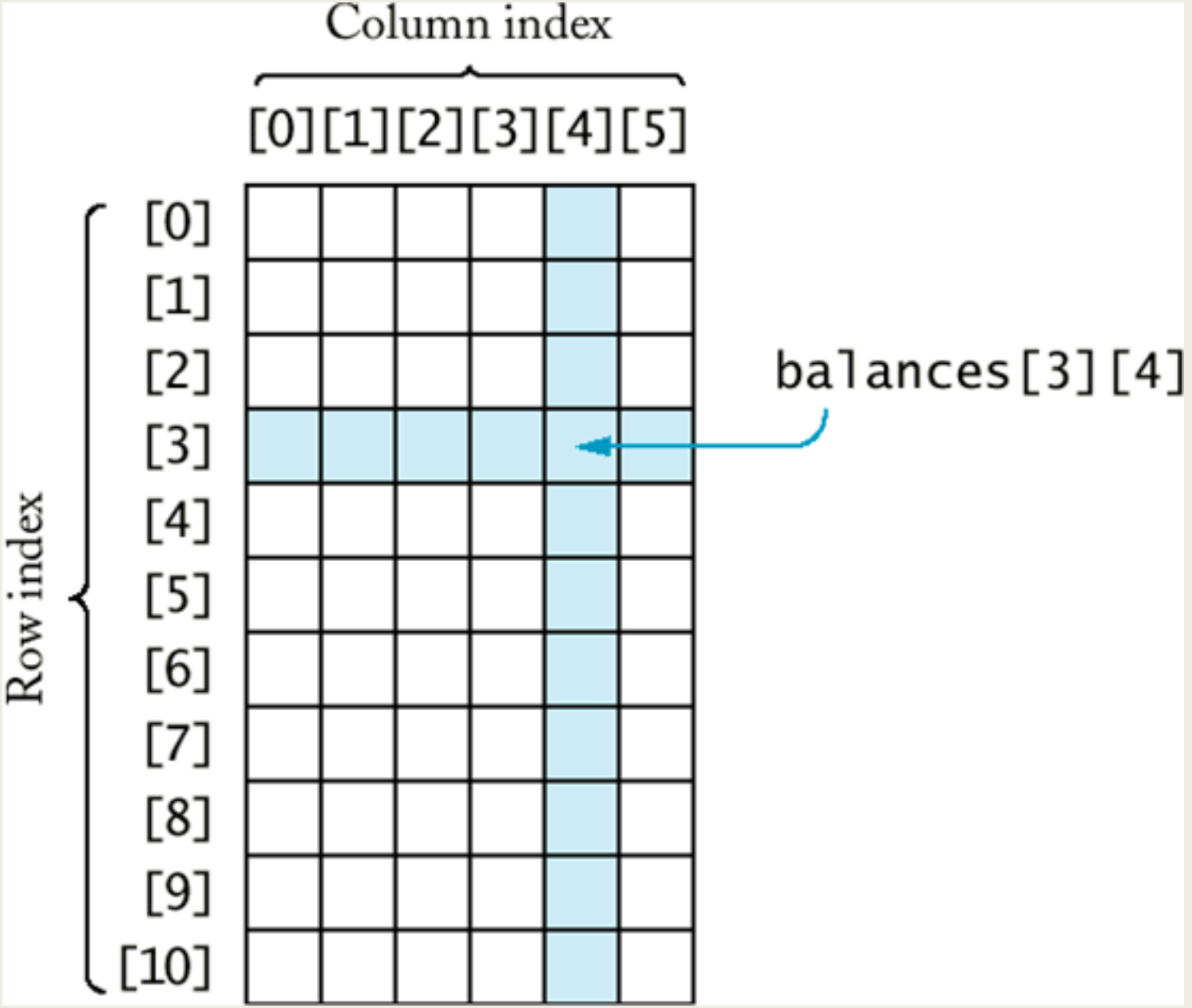



```
int[] array = new int[10];

for (int i = 0; i < size; i++) {
     array[i] = a;
}
```

Call it a static array

Array 2D



Array 2D

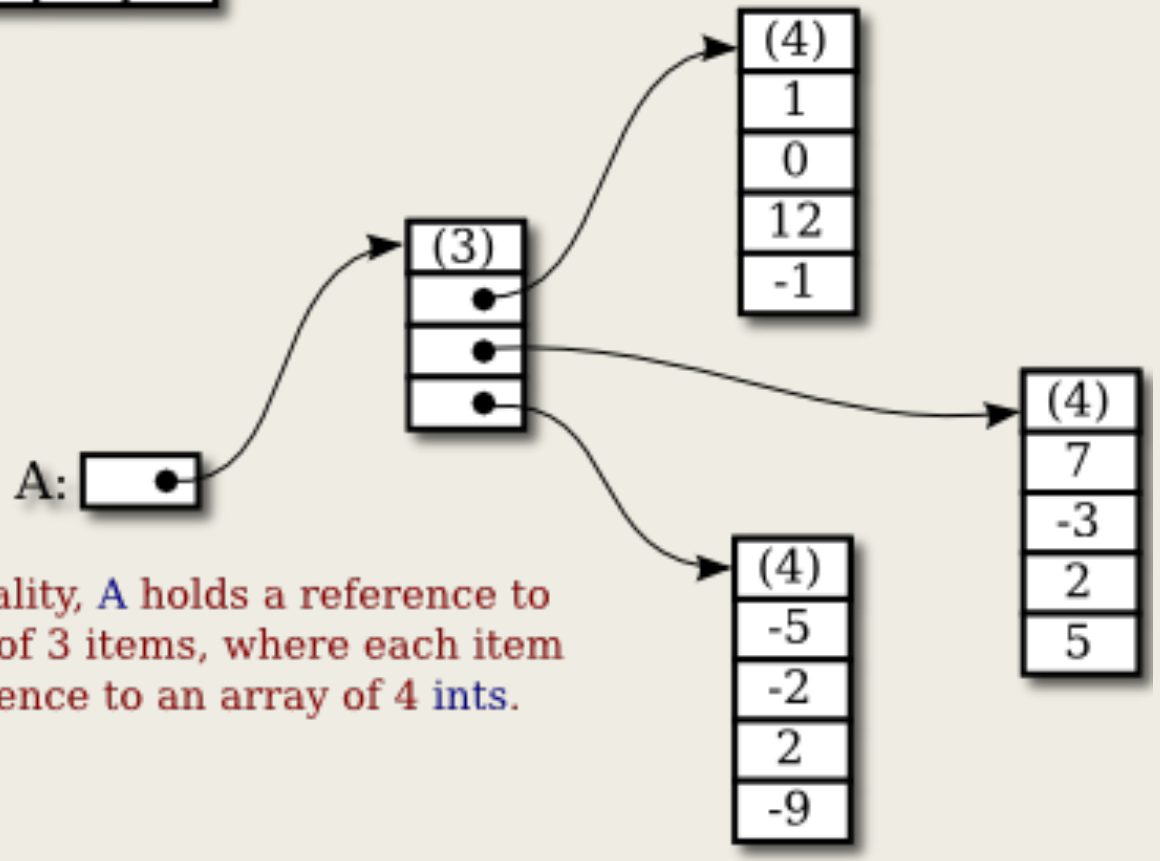
```
int[][] multi = new int[][]{
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }};
```

Array 2D

A:

1	0	12	-1
7	-3	2	5
-5	-2	2	-9

If you create an array A = new int[3][4], you should think of it as a "matrix" with 3 rows and 4 columns.



But in reality, A holds a reference to an array of 3 items, where each item is a reference to an array of 4 ints.