# COMP 250: Java Programming I

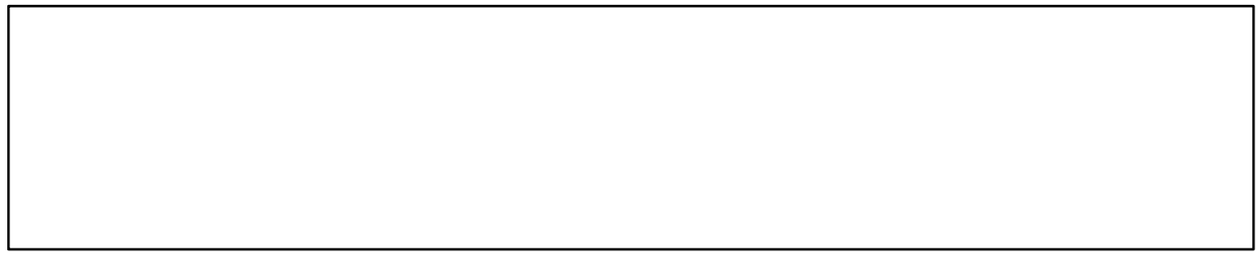
Carlos G. Oliver, Jérôme Waldispühl
January 17-18, 2018
Slides adapted from M. Blanchette

[Downey Ch 2]

# Variables and types

- Variable: temporary storage location in memory. It has
  - a name (to refer to it)
  - a type (to describe what kind of information it can store).
  - a value (content stored in memory)
- Two kinds of types:
  - Primitive types (seen today)
  - Classes (next lecture)

Memory (RAM)



```
public class VariablesExample {

public static void main(String args[]) {
    int age;      // age can store an integer
    float pi;     // float can store a decimal number
     age = 29;
     pi = 3.14;
    }
}
```

[Downey Ch 2]

# Primitive types

| Type | Size | Description | Range |
|---|---|---|---|
| byte | 8-bit | signed integer | [-128,127] |
| char | 16-bit | integer | [0, 65536] (encodes ‘a’,‘b’…) |
| short | 16-bit | signed integer | [-32768,32767] |
| int | 32-bit | signed integer | [-2147483648, 2147383647] |
| long | 64-bit | signed integer | [-9223372036854775807, 9223372036854775806] |
| float | 32-bit | decimal number | 1.40239e-45 to 3.402823e+38 |
| double | 64-bit | decimal number | 4.9406e-324 to 1.79769e+308 |
| boolean | 8-bit | boolean | true or false |

[Downey Ch 2]

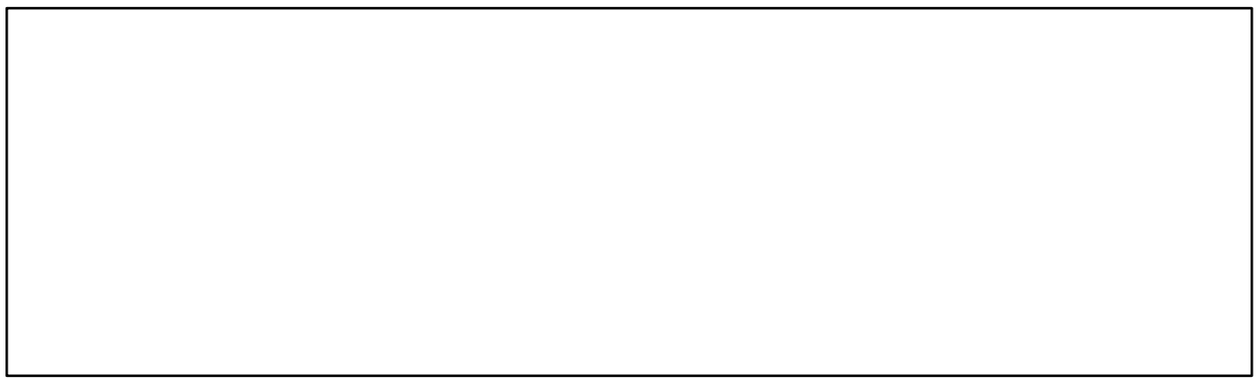
# Expressions and Assignments

- **Expression:** Piece of code that has a value of a certain type
- **Assignment:** Storing the value of an expression into a variable
- **Syntax:** <variable> = <expression>

Examples:

```
public class Expressions {
public static void main(String args[]) {
      int i, j, k;
      float f, g;
      i = 5;
      j = i + 1;
      k = k * 2;  // Compiling error: Why? _____
      j = j / 2;
      j + 10;       // Legal, but useless. Why? _____
      g = j + 3.14; // Note the implicit conversion of j into a float
      f = 15 / 2;    // f now has value 7.0. Why?_____
      f = ((float) i) / 2 // Explicit type conversion (casting) of i into a float. f is now 7.5
}}
```

Memory (RAM)



[Downey Ch 6.5-6.6]

# Boolean expressions

- Boolean expressions have value true or false.
- Operations on booleans:
  - NOT : !
  - AND : &&
  - OR : ||

```
boolean a,b;
int i = 5;
int j = 2;
a = ( i * j < 10 );            // a is _____

b = !a;                        // b is _____
a = (i+j < 10) && (i+j > 0);   // a is ____
b = ( i<0 ) || ( j>1 ) ;       // b is _____
a = ! ( b && ( i>0 || j<i ) ); // a is _____
```

# Exercise

```
public class Exercise {
public static void main(String args[]) {
    int i,j;
    float f;
    boolean a, b;
    char c = 'f';
    f = i;      // compilation error: i is not initialized
    i = 9;
    a = (f > 100);    // compilation error: f is not initialized
    f = i;
    b = true;
    a = ( b || (12345.67*i - f/0.02345 == 0.003464) );
    j = i;
    j = j + 1;     // value of j: 10,   value of i is still 9
    i = f + 3.3;  // error: a float value cannot be stored in an int
    i = (int) (f + 3.3);    // the float value 12.3 is cast into an int.
                            // It becomes 12, so i becomes 12
    b = b && ( (i == j) || (!b || f > 10) );
}}
```

| i | j | f | a | b | c |
|---|---|---|---|---|---|
| - | - | | | | |
| - | - | - | | | |
| - | - | - | - | - | |
| - | - | - | - | - | 'f' |
| 9 | - | - | - | - | 'f' |
| 9 | - | 9.0 | - | - | 'f' |
| 9 | - | 9.0 | - | T | 'f' |
| 9 | - | 9.0 | T | T | 'f' |
| 9 | 9 | 9.0 | T | T | 'f' |
| 9 | 10 | 9.0 | T | T | 'f' |
| 12 | 10 | 9.0 | T | T | 'f' |
| 12 | 10 | 9.0 | T | F | 'f' |

[Downey Ch 4.1-4.4]

# Conditionals

- Syntax: if (<boolean expression>) <statementBlock1>
  [else <statementBlock2>]
- Executes <statementBlock1> only if <boolean expression> is true. Otherwise <statementBlock2> is executed.

```
/* Determines if a point (x,y) is inside a circle of radius r centered at (a,b) */
if ( (a-x)*(a-x) + (b-y)*(b-y) <= r*r ) {
     System.out.println("The point is inside the circle");
     if ( (x==a) && (y==b) ) System.out.println("It is the center");
}
else {
     System.out.println("The point is outside the circle");
      // other statements could be here
}
```

- Note: If the statement block contains a single statement, then the {} can be omitted.

[Downey Ch 7.1-7.3]

# while loops

- Syntax:

  while (<boolean expression>) <statementBlock>

- Keeps executing <statementBlock> repeatedly as long as <boolean expression> is true.

  If <boolean expression> is false from the beginning, then <statement> is never executed.

```
int n = 32;
int i = 0;
int exp = 1;
while (exp < n) {
   i++;          // equivalent to i = i +1;
   exp = exp * 2;  // we could also write exp *= 2;
}
// What is the value of exp and i at the end? _____________________________
```

[Downey Ch 7.1-7.3]

# do-while loops

- do <statementBlock> while (<boolean expression>)
- Same as while-loop but <boolean condition> is checked *after* executing <statementBlock>, so <statement> is always executed at least once.

```
// Keep asking for a price as long as the number entered is not positive
double price = 0;
String line;     // String is a special type of variable. More about strings next week
do {
    System.out.println("Enter price of item:");
    line = stndin.readLine();                    // Read a line from keyboard
    price = Double.parseDouble(line) ;      // Parse the line to get a double
} while (price<=0);
```

[Downey Ch 12.4]

# For loops

- for (<statement1>; <boolean expression>; <statement2>) <statementBlock3>
- Equivalent to:

<statement1>
while (<boolean expression>) {
  <statementBlock3>
  <statement2>
}

What does this print?

_____

```
int n=5;
int s=0;
int i;
for (i = 0; i < n; i++ ) {
    s = s + i;
}
System.out.println(”Value of s:” + s);
```

What does this print?

```
int n=5;
for(int i = 1; i <= n; i++) {
     for (int j = 1; j <= i; j++) {
          System.out.print( i );
   }
   System.out.println(""); // print the end of line
}
```

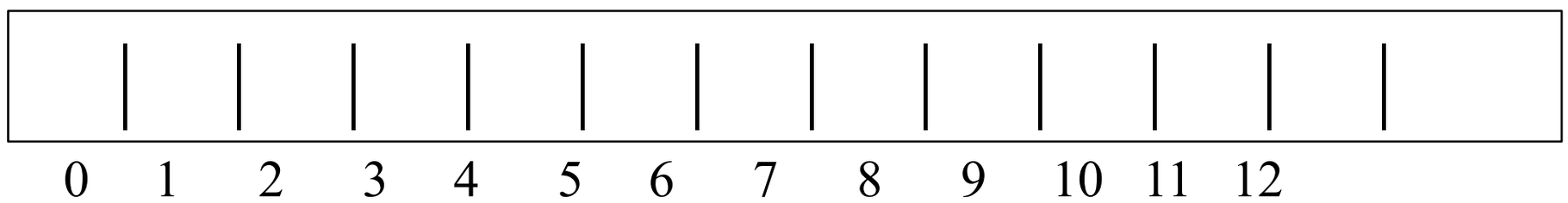
Memory (RAM)

Output

[Downey Ch 12]

# Arrays

- Arrays are used to store and manipulate several variables of the same type
- Array X:




- To access the i-th element:

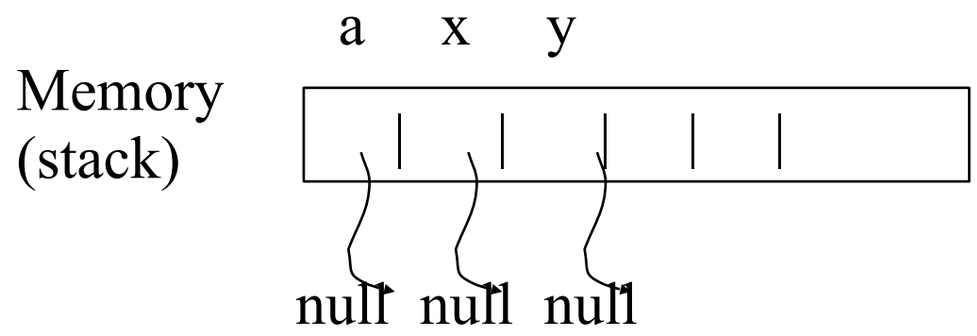
```
i = 5;
X[i] = 2;
int a = X[i] + 1;
```

- Note: first element is at index 0.
- X.length is the number of elements in X (here, it's 13)
- Java makes sure you don't write outside arrays:
  - ArrayIndexOutOfBoundsException gets thrown if you try

[Downey Ch 12]

# Arrays in Java

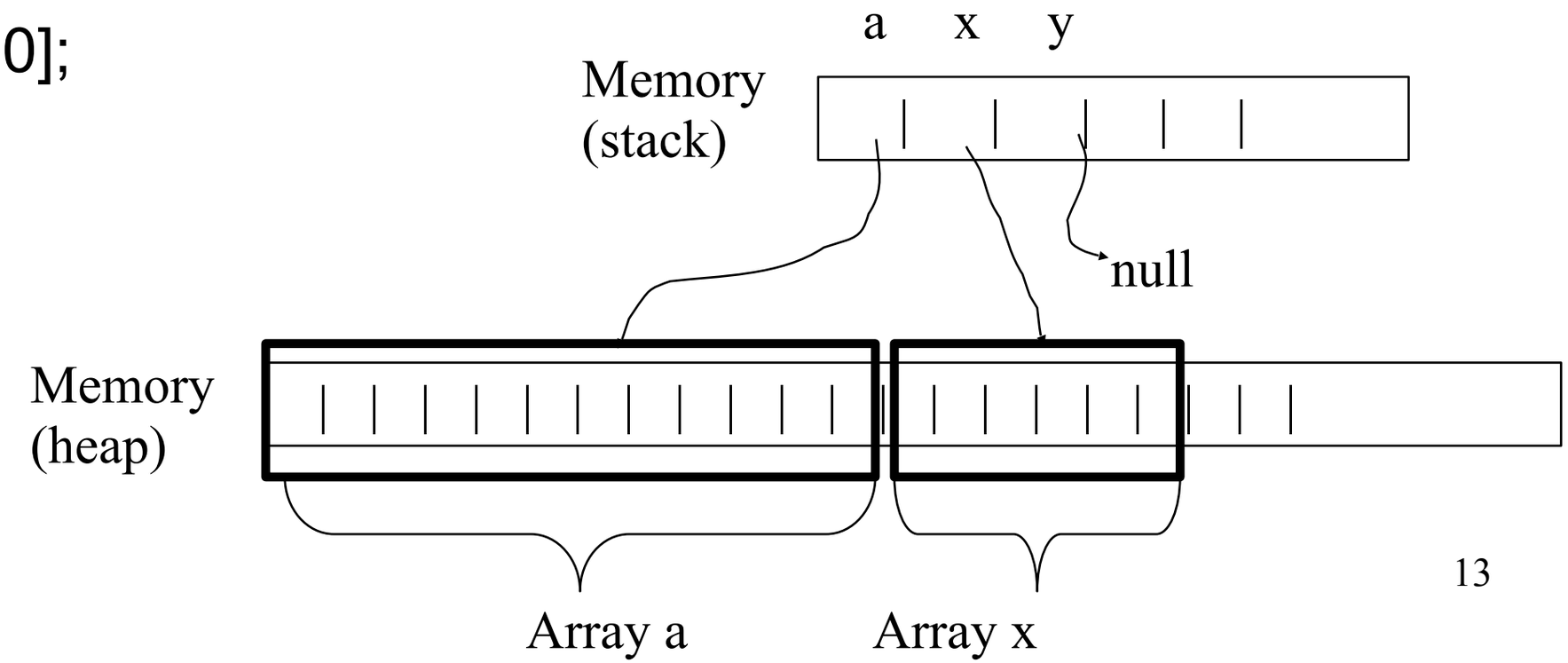
- Declaration:



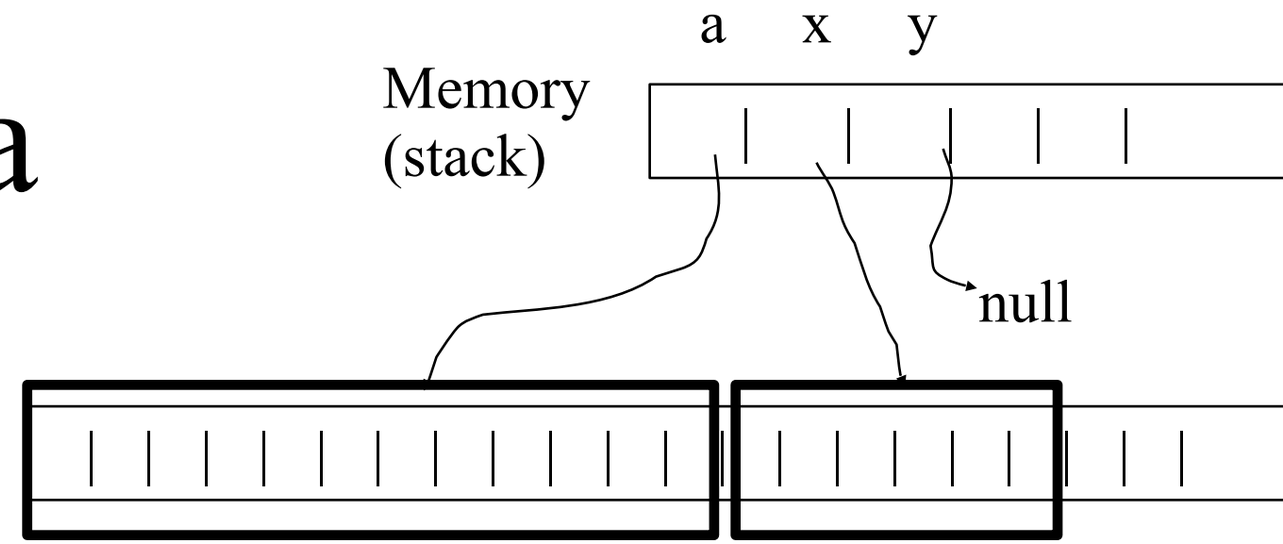

```
double[] a;   // a will be array of double.
int[] x,y;    // x and y will both be arrays of ints
// IMPORTANT: At this point, a, x, and y are references to null arrays
a[ 3 ] = 2;  // would cause error because a is null
```

- Allocation

```
a = new double[10];
x = new int[5];
```



# Arrays in Java

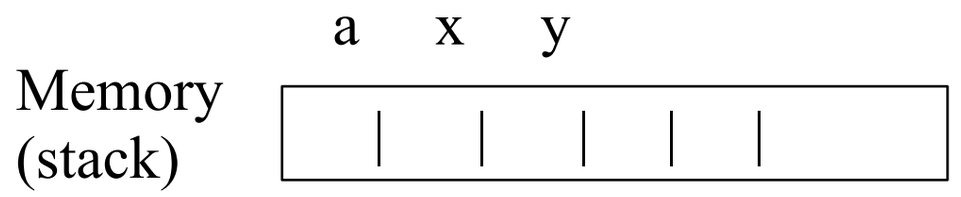


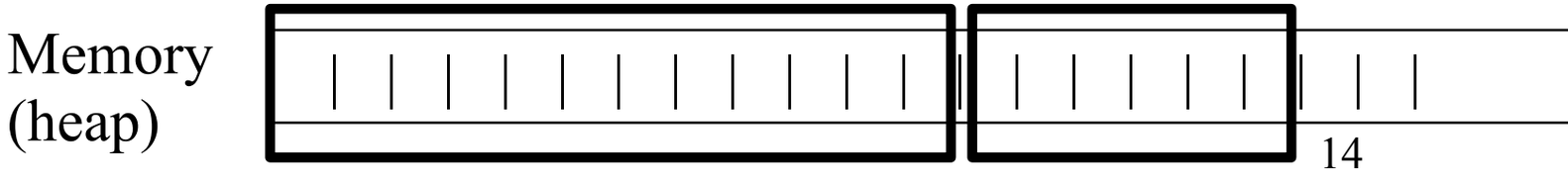

- Accessing elements

```
a[ 3 ] = 3.1;
x[ 0 ] = 1;
a[ x[ 0 ] ] = 1.5;
y[ 3 ] = 2;     // compiling error. Why? _____
x[ a[ 1 ] ] = 3; // compiling error. Why? _____
a[ 10 ] = 3; // run-time error. ArrayIndexOutOfBoundsException gets thrown
```

- Assignments with arrays

```
y=x;
y[ 3 ]=9;       // the value of x[3] is now _____
x[ 2 ]=4;       // the value of y[2] is now _____
a = x;           // compil. error: a and x have different types
x = new int[ y[2] ];
x[3] = 5;    // the value of y[3] is now _____
y = new int[3];
```

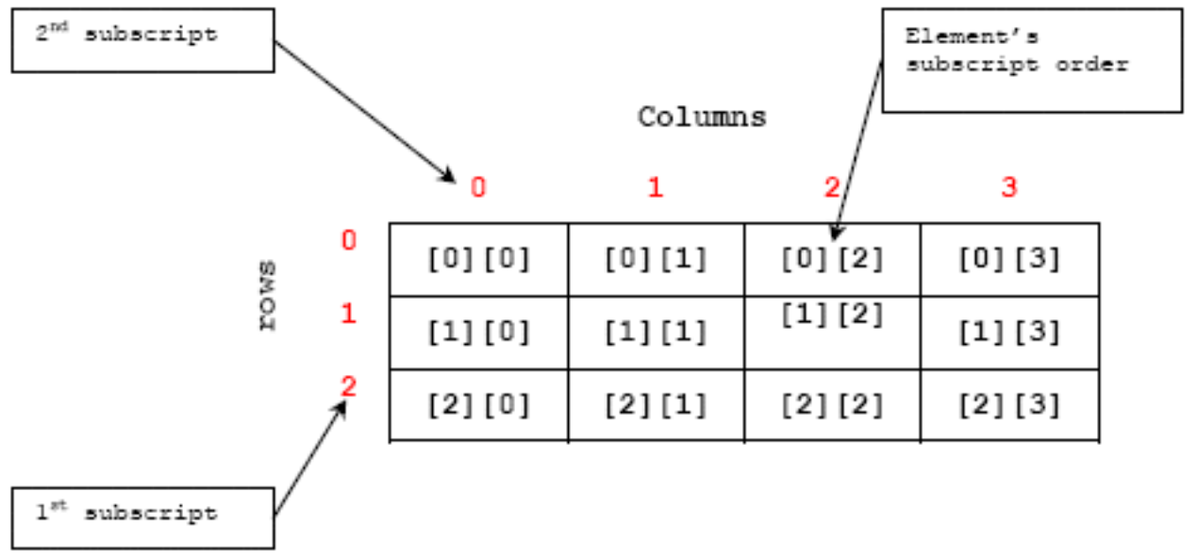


# Multi-dimensional arrays

- Arrays can have more than one dimension:

```
double matrix [ ] [ ] = new double[ 10 ] [ 10 ];
// initialize the matrix to zero
for ( int i=0; i<10; i++ ) {
    for ( int j=0; j<10; j++ ) {
        matrix[ i ][ j ] = 0;
    }
}
```




```
// make it an identity matrix
for ( int i=0; i<10; i++ ) matrix[ i ][ i ]=1;
```

(Image Source: https://i.stack.imgur.com/yFGwe.png)

# Methods

- Method (a.k.a. function, procedure, or routine):
  - Piece of code that carries a specific computation
  - Can be called (executed) from anywhere in the code (if they are public)
  - Can take one or more parameters (arguments) as input
  - Can return a value (or an array, or any object)

```
public static float square( float x ) {
    float s = x*x;
    return s;
}
```

- Local variables:
  - Variables declared inside a method (e.g. s).
  - They are discarded after the method finishes being executed.

```
public class Course{
  // prints welcoming statement. Takes no arguments. Returns nothing
 public static void printWelcome() {
      System.out.println("Welcome to COMP 250");
   }

   // prints welcoming statement for the given courseID. Returns nothing
   public static void printWelcome(int courseID) {
      System.out.println("Welcome to COMP ” + courseID);
   }

   // returns the letter grade for the given percent grade
   public static char getGradeFromPercent( double percent) {
      char grade;
      if ( percent >= 0.8 ) grade =  'A';
      if ( percent >= 0.7 && percent < 0.8 ) grade =  'B';
       if (percent < 0.7) grade = ‘C’;
      return grade;
   }

   public static void main (String args[]) {
      printWelcome();
      printWelcome( 203 );
      char g = getGradeFromPercent(0.67);
      System.out.println("The grade is " + g);
      grade = ‘A’;      // compilation error: ‘grade’ was only defined inside                    //
     getGadeFromPercent method
   }
}
```

# Why are methods useful?

- **Code re-use**: a method can be called (executed) as often as we want, from anywhere in the program. No need to duplicate code.
- **Encapsulation**: Allows to think of a piece of code as a black box with a well-defined function. Users don't need to know *how* the method works, only *what* the method does: what are its arguments, what does it return.
- **Makes program much easier to design, understand and debug**

# Parameter passing

```
// Returns area a circle of radius r
static double circleArea(double r) {
    double a = 3.1416 * r * r ;
     r = -1 ;  // just to see what happens
    return a ;
}

public static void main(String args[]) {
  double radius = 2;
  double area = circleArea(radius);
  System.out.println("Radius:”
            + radius + " Area: ” + area );
}

Output:  Radius: 2 Area:12.5664
```

Memory (stack)

# The truth about parameter passing

- What happens when a method is called?
  1. The flow of execution of the code calling the method is interrupted.
  2. If the methods takes some arguments, these arguments are allocated in memory (stack). They are initialized with the value of the arguments provided by the caller.
  3. If variables are declared within the method, they are also put on the stack.
  4. The code of the method is executed. This may include calling other methods.
  5. When the code of the method has been executed, it may return a value to the caller. All local variables and arguments created on the stack are discarded.
- Summary: Parameters are passed by value
  - The method called receives a *copy* of the parameters passed
  - Since it is working a copy, the method *can't change the original*
  - But watch out with arrays and non-primitive types...

```
static void stupidIncrement( int a ) {
    int i = a;
    i = i + 1;
    System.out.println("In stupidIncrement, i = " + i);
}
static void fakeAssign( int a, int b) {
    a = b;
    System.out.println("In fakeAssign, a = " + a + " and b = " + b );
}
static int add(int a, int b) {
    int sum = a + b;
    a = 0;
    return sum;
}
static public void main(String args[]) {
    int a = 1, b = 2, i = 9;
    fakeAssign( a, b );
    System.out.println("After fakeAssign a:" + a + "    b: " + b + "      i:" + i);
    stupidIncrement(b);
    System.out.println("After stupid a:" + a + "     b:" + b + "     i:" + i);
    stupidIncrement(i);
    System.out.println("Again after stupid a:" + a + "     b:" + b + "     i:" + i);
    a = add(i, a);
    System.out.println("After add a:" + a + "        b:" + b + "       i:" + i);
    System.out.println("sum = " + sum);   // this causes an compilation error
}                                          // because sum is only defined inside "add"
```

Output:

In assign, a =2  and b =  2
AfterfakeAssign  a:1    b:2   c:9   // because in fakeAssign, we were working
// only on copies of the original a and b
In stupidIncrement, i = 3
After stupidIncrement, a: 1      b: 2     i: 9     // the variable i used in                                    //
fakeAssign has nothing to do
                                    // with the variable i defined in main
In stupidIncrement, i = 10
Again after stupidIncrement a: 1      b: 2     i: 9
After add a: 10     b: 2      i: 9

# Parameter passing with arrays

```
static void changeArray( int a[] ) {
    System.out.println(“First, a[0] is “ + a[0]);
    a[0]=2;
    System.out.println(”Then, a[0] is " + a[0]);
    a = new int[2];
    a[0]=3;
    System.out.println("Then, a[0] is " + a[0]);
}
public static void main(String args[]) {
    int[] array;
    array = new int[3];
    array[0] = 1;
    changeArray(array);
    System.out.println("Finally, array[0] is " + array[0] );
}
```

# Parameter passing with arrays

- Be careful with arrays and other reference types!
- When you pass a reference to a variable to a method, the method can use that reference to modify the value stored in memory.