3: Loops, Arrays
Foo Corporation needs a program to calculate how much to pay their employees.

1. Pay = hours worked x base pay
2. Hours over 40 get paid 1.5 the base pay
3. The base pay must be no less than $8.00
4. The number of hours must be no more than 60
Frequent Issues (I)

The signature of the main method cannot be modified.

```java
public static void main(String[] arguments) {
    ...
}
```
Frequent Issues (II)

Return values: if you declare that the method is not `void`, then it has to return something!

```java
public static int pay(double basePay, int hours) {
    if (basePay < 8.0)
        return -1;
    else if (hours > 60)
        return -1;
    else {
        int salary = 0;
        ...
        return salary;
    }
}
```
Frequent Issues (III)

Don't create duplicate variables with the same name

```java
public static int pay(double basePay, int hours) {
    int salary = 0; // OK
    int salary = 0; // salary already defined!!
    int salary = 0; // salary already defined!!
}
```
Frequent Issues (III)

class WeeklyPay {
    public static void pay(double basePay, int hours) {
        if (basePay < 8.0) {
            System.out.println("You must be paid at least $8.00/hour");
        } else if (hours > 60) {
            System.out.println("You can't work more than 60 hours a week");
        } else {
            int overtimeHours = 0;
            if (hours > 40) {
                overtimeHours = hours - 40;
                hours = 40;
            }
            double pay = basePay * hours;
            pay += overtimeHours * basePay * 1.5;
            System.out.println("Pay this employee $" + pay);
        }
    }
    public static void main(String[] arguments) {
        pay(7.5, 35);
        pay(8.2, 47);
        pay(10.0, 73);
    }
}
What we have learned so far

- Variables & types
- Operators
- Type conversions & casting
- Methods & parameters
- If statement
Today’s Topics

- Good programming style
- Loops
- Arrays
Good Programming Style
Good programming style

The goal of good style is to make your code more readable.

By you and by others.
Rule #1: use good (meaningful) names

String a1;
int a2;
double b; // BAD!!

String firstName; // GOOD
String lastName; // GOOD
Int temperature; // GOOD
Rule #2: Use indentation

```java
public class test {
    public static void main (String[] arguments) {
        int x = 5;
        x = x * x;
        if (x > 20) {
            System.out.println(x + " is greater than 20.");
        }
        double y = 3.4;
    }
}
```

```java
public class test {
    public static void main(String[] arguments) {
        int x = 5;
        x = x * x;
        if (x > 20) {
            System.out.println(x + " is greater than 20.");
        }
        double y = 3.4;
    }
}
```
Rule #3: Use whitespaces

Put whitespaces in complex expressions

```c
// BAD!!
double cel = fahr*42.0/(13.0-7.0);

// GOOD
double cel = fahr * 42.0 / (13.0 - 7.0);
```
Rule #4: Use whitespaces

Put blank lines to improve readability:

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

    int x = 5; x = x * x;
    if (x > 20) {
        System.out.println(x + " is > 20.");
    }

    double y = 3.4;
}
```
Rule #5: Do not duplicate tests

```java
if (basePay < 8.0) {
    ...
}
else if (hours > 60) {
    ...
}
else if (basePay >= 8.0 && hours <= 60) {
    ...
}
```
Rule #5: Do not duplicate tests

```java
if (basePay < 8.0) {
    ...
}
else
    if (hours > 60) {
        ...
    }
else
    if (basePay >= 8.0 && hours <= 60) {
        ...
    }
```
Use good names for variables and methods
Use indentation
Add whitespaces
Don't duplicate tests
Loops
Loops

What if you want to do it for 200 Rules?

```java
static void main (String[] arguments) {
    System.out.println("Rule #1");
    System.out.println("Rule #2");
    System.out.println("Rule #3");
}
```
Loops

Loop operators allow to loop through a block of code.

There are several loop operators in Java.
The *while* operator

Allows a programmer to state that an action (or a block of them) will be executed as long as certain condition is met

```
while (condition){
    STATEMENTS;
}
```

Must be a boolean expression
The **while** operator

```java
int i = 0;
while (i < 3) {
    System.out.println("Rule #" + i);
    i = i+1;
}
```

Count carefully

Make sure that your loop has a chance to finish

- Meeting the condition has to be closer as the number of iterations grows
The *for* operator

Execute an statement (or block of them) a given number of times

```plaintext
for (initialization; condition; update) {
    STATEMENTS;
}
```
The **for** operator

```java
for (int i = 0; i < 3; i = i + 1)
    System.out.println("Rule "+ i);
```

- **Condition** is a boolean expression, which is computed at the end of each iteration. If it yields true, another iteration comes.

- The initialization expression marks the start of the loop. In general, it consists of declaring and initializing a variable so-called **control variable**.

- The update expression is executed at the end of each iteration. In general, it consists of increasing the control variable.

*i = i+1 may be replaced by i++*
– Print all the integers between 1 and 20

```
for(int i = 1; i <= 10; i++)
{
    System.out.println(i);
}
```

– Print all the even numbers between 20 and 2

```
for(int i = 20; i >=0; i -= 2)
{
    System.out.println(i);
}
```
Branching Statements

One might want to leave the loop, even though the condition has not been met yet

- `break` terminates a `for` or `while` loop

```java
for (int i=0; i<100; i++) {
    if(i == 50)
        break;
    System.out.println("Rule #" + i);
}
```
Branching Statements

One might want to leave the current statement and go directly to the next one

- `continue` skips the current iteration of a loop and proceeds directly to the next iteration

```java
for (int i=0; i<100; i++) {
    if(i == 50)
        continue;
    System.out.println("Rule #" + i);
}
```
Embedded loops

Scope of the variable defined in the initialization (control variable): respective for block

```java
for (int i = 0; i < 3; i++) {
    for (int j = 2; j < 4; j++) {
        System.out.println (i + " " + j);
    }
}
```
do - while

Regarding while loops, the statements are always executed at least once

- Since condition is not computed until the end of the first iteration

```c
do {
    STATEMENTS;
} while (condicion);
```
do - while

Example

- Write down numbers between 1 and 10;

```java
int i= 1;
do {
    System.out.println(i);
i++;
} while (i <= 10);
```
Tips

Just while loop ends with ‘;’

```java
for (int i=0; i<10; i++) {
    System.out.println("i is " + i);
}

BAD

int i=0;
while (i < 10) {
    System.out.println("i is " + i);
    i++;
}

GOOD

int i=0;
do {
    System.out.println("i is " + i);
    i++;
} while (i<10);
```
Arrays
Arrays

An array is an indexed list of values

- You can make an array of any type (int, double, String, etc.)
- All elements of an array must have the same type
- We can refer to the whole list of values (the array variable) …
- ... or to one specific value
- We can, as well, modify the list of values by adding, deleting of modifying each specific value
Arrays

The first element of the array is located at index 0, while the last one is located at index n - 1.
Arrays

The first element of the array is located at index 0, while the last one is located at index $n - 1$

- Example: double [];

```
5.0  2.44  9.01  1.0
```

```
-9.9
```

$$0 \quad 1 \quad 2 \quad 3$$

$$n - 1$$
Arrays

Array definition
– TYPE []

Arrays are just another type
– Arrays of array type can be defined

```c
int [] values;  // array of int values
int [][] values;
// array of array of int values
```

int [] is a data type
Arrays

To create an array of a given size, use the \texttt{new} operator

– Or you may use a variable to specify the size:

\begin{verbatim}
int[] values = new int[5];

// using a variable
int size = 12;
int[] values = new int[size];
\end{verbatim}
Array Initialization

Curly braces can be used to initialize an array.

- It can **ONLY** be used when you declare the variable.

```java
int[] values = {12, 24, -23, 47};
```

<table>
<thead>
<tr>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
</tr>
<tr>
<td>24</td>
</tr>
<tr>
<td>-23</td>
</tr>
<tr>
<td>47</td>
</tr>
</tbody>
</table>

The size of the array is implicitly set to 4
Arrays

The first element of the array is located at index 0, while the last one is located at index n - 1

```java
int[] values = new int[10];
values[0] = 10; // CORRECT
values[1] = 11; // CORRECT
values[2] = 12; // CORRECT
values[3] = 13; // CORRECT
values[9] = 19; // CORRECT
values[10] = 20; // WRONG!!
// compiles but throws an Exception
// at run-time (demo)
```

<table>
<thead>
<tr>
<th>values</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>...</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>...</td>
<td>n - 1</td>
</tr>
</tbody>
</table>
Quiz time!

Is there an error in this code?

```c
int[] values = {1, 2.5, 3, 3.5, 4}
```
Accesing Arrays

To access the elements of an array:

– Use the [] operator and state the position needed

```java
int[] values = {12, 24, -23, 47};
values[3] = 18; // {12, 24, -23, 18}
int x = values[1] + 3; // {12, 24, -23, 18}
```

Array starts at position 0 and ends at position length - 1
Accessing Arrays

To access the elements of an array, use the [] operator:

```
values[index]
```

Example:

```java
int[] values = { 12, 24, -23, 47 };
values[3] = 18;       // {12, 24, -23, 18}
int x = values[1] + 3; // {12, 24, -23, 18}
```
The *length* variable

Each array has a *length* variable built-in that contains the length of the array.

```java
int[] values = new int[12];
int size = values.length; // size = 12

int[] values2 = {1,2,3,4,5};
int size2 = values2.length; // size = 5
```
public static void main (String[] arguments) {
    System.out.println(arguments.length);
    System.out.println(arguments[0]);
    System.out.println(arguments[1]);
}

String arrays

A side note
Using Arrays
Arrays as arguments

// method to print an Array
public static void printArray(int[] array) {
    for (int i = 0; i < array.length; i++) {
        System.out.print(array[i] + " ");
    }
}
(...)

// method call
int[] list = {3, 1, 2, 6, 4, 2};
printArray(list);

// method call (another shape of)
printArray(new int[]{3, 1, 2, 6, 4, 2});
Arrays

Arrays as method output

```java
// Array inversion method
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    // declaring the array to be returned

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }

    return result; // devolvemos el array
}
(...) // method call
int[] list1 = new int[]{1, 2, 3, 4, 5, 6}; // array to invert
int[] list2 = reverse(list1); // inverted array
```
Arrays

Array Utils

– Arrays copy $\rightarrow$ System.arraycopy(...)
Arrays

Array Utils (II)

– java.util.Arrays

public static char[] copyOfRange(char[] original, int from, int to)

class ArrayCopyDemo {
    public static void main(String[] args) {
        char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e', 'i', 'n', 'a', 't', 'e', 'd' };
        char[] copyTo = java.util.Arrays.copyOfRange(copyFrom, 2, 9);
        // no necesitamos crear el Array
        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        // salida: caffein
        System.out.println(Arrays.toString(copyTo));
        // salida: [c, a, f, f, e, i, n]
    }
}
Arrays

Array Utils (III)

- `java.util.Arrays`

```java
int binarySearch(tipo[] a, tipo key)
// returns the position of 'key' in 'a' array

boolean equals(tipo[] a, tipo[] a2)
// yields true if 'a' and 'a2' contain the same values

void fill(tipo[] a, tipo val)
// set every position of array 'a' to 'val'

void sort(tipo[] a)
// orders 'a' array (ASC)
```
Combining Loops and Arrays
Looping through an array

Example 1: iterating over an array (for)

```java
int[] values = new int[5];

for (int i=0; i < valores.length; i++) {
    valores[i] = i;
    int y = valores[i] * valores[i];
    System.out.println(y);
}
```
Looping through an array

Example 2: iterating over an array

```java
int[] valores = new int[5];
int i = 0;
while (i < valores.length) {
    valores[i] = i;
    int y = valores[i] * valores[i];
    System.out.println(y);
    i++;
}
```

Provided we are going to iterate over an array, a for loop seems more appropriate.
Looping through an array

Iterando an array (improved for)

```java
for(tipo variable_iteración: array)
    instrucciones;

int[] valores = {1,2,3,4,5};
int suma = 0;

for (int x: valores) {
    suma += x; // suma = suma + valores[i]
    System.out.println(suma);
}
```

Avoiding control variables, and array limits issues
Iterando an array (improved for)

- Can leave the loop using the break statement

```java
int[] valores = {1,2,3,4,5};
int suma = 0;

for (int x: valores) {
    suma += x; // suma = suma + valores[i]
    System.out.println(suma);
    if (suma > 100)
        break;
}
```

- But cannot modify the array

  - x = ...
Summary for today

1. Programming Style

2. Loops

3. Arrays
Assignment

A group of friends participate in the Boston Marathon.

Find the best performer.

Find the second-best performer.
3: Loops, Arrays