6.092: Intro to Java

3: Loops, Arrays

Assigment 2

- Calculate employee salaries for Foo Corp.

1. Pay = hours worked x base pay
2. Hours over 40 get paid 1.5 the base pay
3. The base pay >= $8.00
4. The number of hours <= 60

Popular Problems 1

- Signature of the *main* method *cannot* be modified.

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

Popular Problems 2

- If method return type != *void*, it must return a value!

```
public static int pay(double basePay, int hours) {
    if (basePay < 8.0)   return -1;
    else if (hours > 60) return -1;

}
```

What if code comes here?

Popular Problems 2

- If method return type != *void*, it must return a value!

```
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
    }
}
```

Popular Problems 3

- Duplicate variables with same name

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

```java
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

Goal: readable code

By you and by others.

Rule #1: Use Meaningful Names

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

String firstName; // GOOD
String lastName;  // GOOD
int temperature;  // GOOD
```

Rule #2: Use indentation

```
public static void main (String[] arguments) {
 int x=5;x =x* x;
if(x >20) System.out.println(x +
“> 20.”);  double y = 3.4;}
```

Ctrl-shift-F to auto-format the file

Rule #2: Use indentation

```
public static void main (String[] arguments) {
    int x = 5;
    x = x * x;
    if (x > 20) {
        System.out.println(x + “> 20.”);
    }
    double y = 3.4;
}
```

Ctrl-shift-F to auto-format the file

Rule #3: Use Whitespaces

- Whitespaces in complex expressions

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

```
// GOOD
double cel = fahr * 42.0 / (13.0 - 7.0);
```

Rule #3: Use Whitespaces

- Use blank lines to separate blocks

```
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 #4: Don’t Duplicate Tests

```
if (basePay < 8.0) {
            ...
} else if (hours > 60) {
            ...
} else if (basePay >= 8.0 && hours <= 60){
            ...
}
```

BAD

Rule #4: Don’t Duplicate Tests

```
if (basePay < 8.0) {
            ...
} else if (hours > 60) {
            ...
} else {
            ...
}
```

Good

Summary

- Use good variable/method names
- Use indentation
- Add whitespaces
- Don't duplicate tests

Loops

Loops

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

What if you want to do it for 200 Rules?

Loops

- Loop through a block of code.

- Several loop operators in Java.
  - while
  - for

The *while* operator

```
while (condition) {
    statements
}
```

The *while* operator

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

- Count carefully (off-by-one error)
- Make sure your loop will finish
  - `while (true);`

The *for* operator

```
for (initialization; condition; update){
    statements
}
```

The *for* operator

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

Note: `i = i+1` may be replaced by `i++`

Branching Statements

break terminates a *for* or *while* loop

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

Branching Statements

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

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

A Loop Within a Loop

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

Variable defined in initialization can be used within its *for* block

Arrays

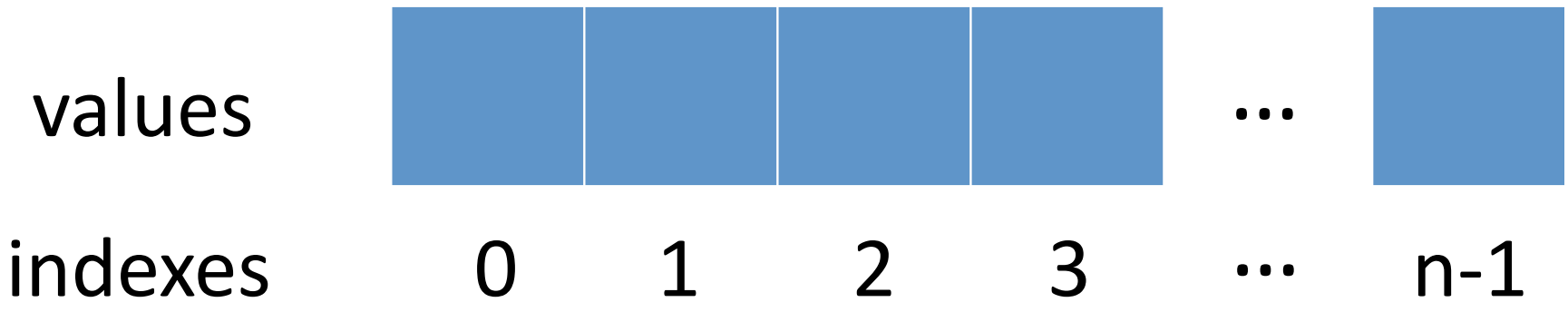
Arrays

- An array is an indexed list of values.

- You can make an array of any type
  - `int, double, String, etc..`

- All elements of array *must* have the same type.

Arrays Example



Arrays Example

```
double[] arr;
```

values	5.0	2.4	11.9	-22.0	…	2.0
indexes	0	1	2	3	…	n-1

Defining Arrays

An array is defined using TYPE [ ]

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

Arrays are just another type.

```
int[][] values;   // int[] is a type
                  // (int[])[] values;
```

Creating Arrays

To create an array of a given size, use operator new:

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

or you may use a variable to specify the size:

```
int size = 12;
int[] values = new int[size];
```

Initializing Arrays

Curly braces can be used to initialize an array.

It can ONLY be used when you declare the variable.

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

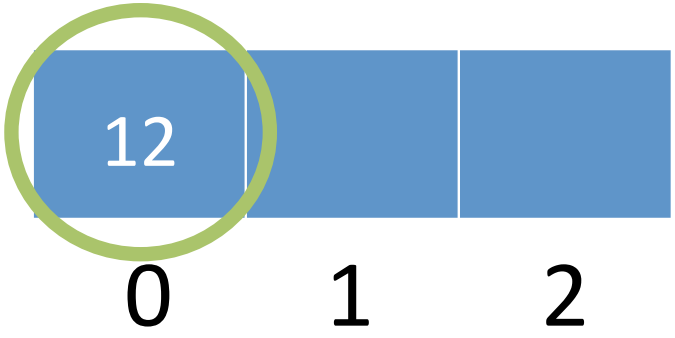
Quiz time!

Is there an error in this code?

```
int[] values;
values = {1, 2.5, 3, 3.5, 4};
```

Accessing Arrays (1)

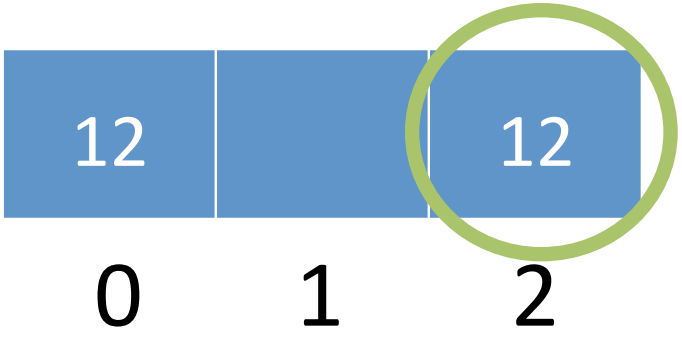
- The index starts at zero and ends at length-1.




```
int[] values = new int[3];
values[0] = 12; // CORRECT
```

Accessing Arrays (1)

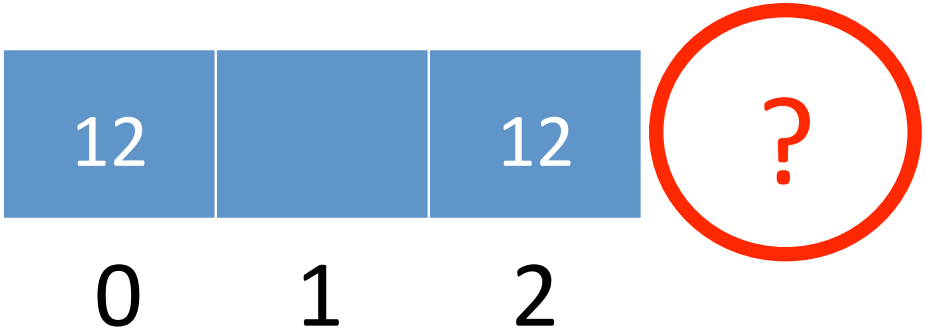
- The index starts at zero and ends at length-1.




```
int[] values = new int[3];
values[0] = 12; // CORRECT
values[2] = 12; // CORRECT
```

Accessing Arrays (1)

- The index starts at zero and ends at length-1.




```
int[] values = new int[3];
values[0] = 12; // CORRECT
values[2] = 12; // CORRECT
values[3] = 12; // WRONG!! compiles but throws an
                // Exception at run-time
```

Accessing Arrays (2)

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

0	5	10	15
0	1	2	3

```
int[] values  = { 0, 5, 10, 15 };
```

Accessing Arrays (2)

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

0	5	10	**18**
0	1	2	3

```
int[] values  = { 0, 5, 10, 15 };
values[3] = 18;
```

Accessing Arrays (2)

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

0	5	10	18
0	1	2	3

```
int[] values  = { 0, 5, 10, 15 };
values[3] = 18;
int x = values[1] + 3;
```

The *length* variable

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

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

int[] values2 = {1,2,3,4,5}
int size2 =     values2.length; // 5
```

String arrays

A side note

```
public static void main (String[] args){
    System.out.println(args.length);
    System.out.println(args[0]);
    System.out.println(args[1]);
}
```

Combining Loops and Arrays

Print square of elements in `values`

```
int[] values = {1,2,3,5,7};
```

Array looping using *for*

```
int[] values = {1,2,3,5,7};
int square = 0;

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

Array looping using *while*

```
int[] values = {1,2,3,5,7};
int i = 0;
int square = 0;
while (i < values.length) {
   square = values[i] * values[i];
   System.out.println(square);
   i++;
}
```

Enhanced *for* loop

Feature in J2SE 5.0 to iterate through values in an array

```
for (int i : values) {// for each int in values
    System.out.println(i);
}
```

The same as:

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

Today’s Summary

1. Programming Style
2. Loops
3. Arrays

Assignment 3

A group of friends participate in the Boston Marathon.

Find the best performer.

Find the second-best performer.