

MIT OpenCourseWare
<http://ocw.mit.edu>

6.092 Introduction to Software Engineering in Java
January (IAP) 2009

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.

6.092: Intro to Java

3: Loops, Arrays

Assignment 2

Foo Corporation needs a program to calculate how much to pay their employees.

1. $\text{Pay} = \text{hours worked} \times \text{base pay}$
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.

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

Frequent Issues (II)

Conditions in *if* statements

Use *else* to avoid duplicate operations

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

Use conjunction (&&) and disjunction (||) of conditions

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

Frequent Issues (III)

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

```
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 (IV)

Don't create duplicate variables with the 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!!  
    ...  
}
```

```
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

What we are going to learn today...

“*Good*” programming style

Loops

Arrays

Good programming style

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

By you and by others.

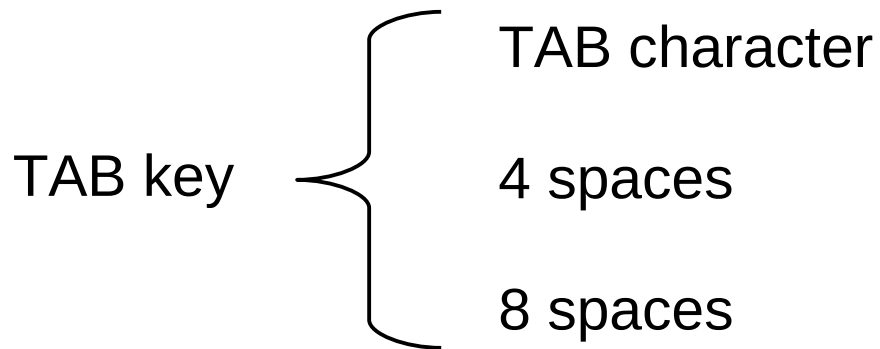
Rule #1: use good (meaningful) names

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

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

Rule #2: Use indentation

```
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;  
}
```



(depending on your editor;
Ctrl-I to auto-indent a block in Eclipse)

Rule #3: Use whitespaces

Put whitespaces in complex expressions:

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

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

Put blank lines to improve readability:

```
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 #4: Do not duplicate tests

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

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

Good programming style (summary)

Use good names for variables and methods

Use indentation

Add whitespaces

Don't duplicate tests

Loops

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

What if you want to do it for 200 lines?

Loops

Loop operators allow to loop through a block of code.

There are several loop operators in Java.

The *while* operator

```
while (condition) {  
    statement  
}
```

The *while* operator

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

Count carefully

Make sure that your loop has a chance to finish.

The *for* operator

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

The *for* operator

```
int i;
```

```
for (i = 0; i < 3; i=i+1) {  
    System.out.println ("This is line " + i);  
}
```

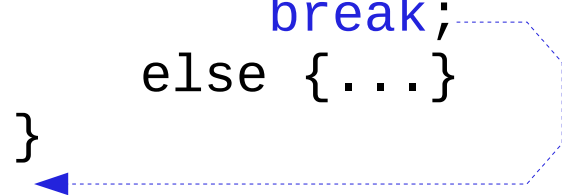
Note: `i = i+1` may be replaced by `i++`

```
for (int i=0; i<3; i++) {  
    ...  
}
```

Branching Statements

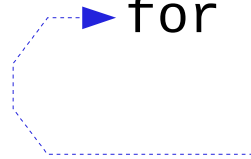
break terminates a *for* or *while* loop

```
for (int i=0; i<100; i++) {  
    if(i == terminationValue)  
        break;  
    else {...}  
}
```



continue skips the current iteration of a *for* or *while* loop and proceeds directly to the next iteration

```
for (int i=0; i<100; i++) {  
    if(i == skipValue)  
        continue;  
    else {...}  
}
```



Embedded loops

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

Scope of the variable defined in the initialization: respective *for* block

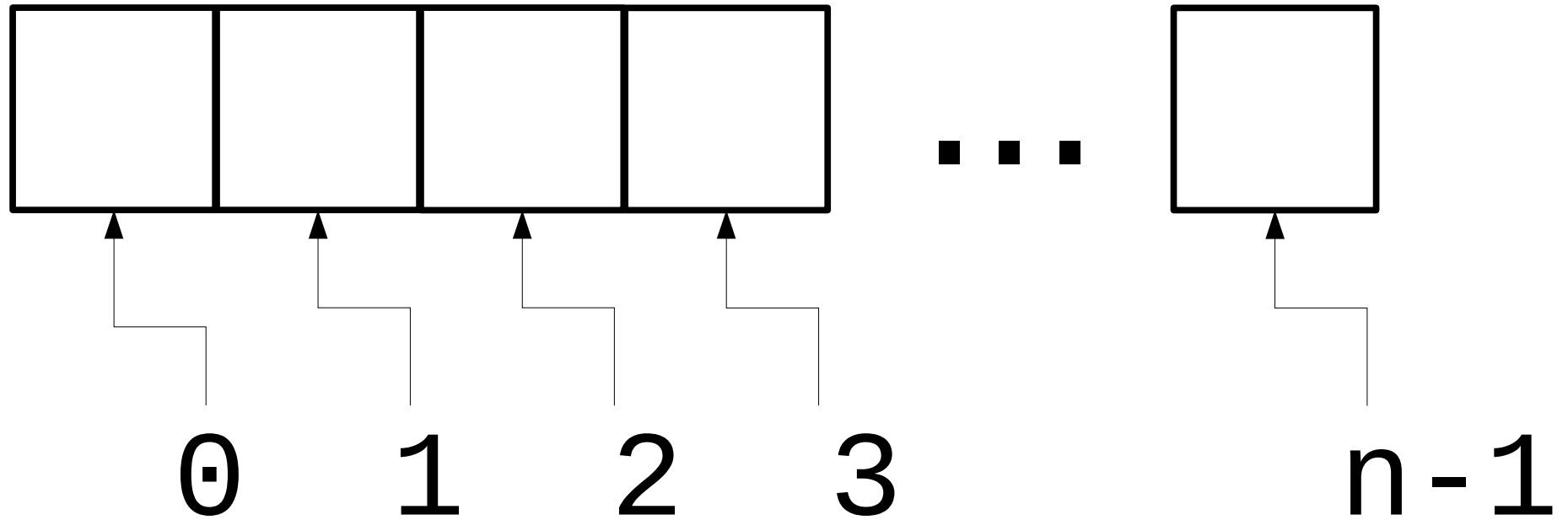
Arrays

An array is an indexed list of values.

You can make an array of int, of double, of String, etc.

All elements of an array must have the same type.

Arrays



Arrays

An array is noted using `[]` and is declared in the usual way:

```
int values[]; // empty array  
int[] values; // equivalent declaration
```

To create an array of a given size, use the 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];
```

Array Initialization

Curly braces can be used to initialize an array in the array declaration statement (and only there).

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

Arrays

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

`values[index]`

Example:

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

```
values[3] = 18;    // write
```

```
int x = values[1] + 3; // read
```

Arrays

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

Example:

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

```
values[0] = 12; // CORRECT
```

```
values[4] = 12; // CORRECT
```

```
values[5] = 12; // WRONG!! compiles but throws  
                // an Exception at run-time
```

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
```

Arrays of String

```
public static void main (String[] arguments){  
    System.out.println("Nbre arguments : " +  
                        arguments.length);  
    System.out.println(arguments[0]);  
    System.out.println(arguments[1]);  
}
```

Looping through an array

Example 1:

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

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


Looping through an array

Example 2:

```
double values[] = new double[25];
```

```
int j=0;
```

```
while (j<values.length) {
```

```
    . . .  
    j++;
```

```
}
```

Enhanced *for* loop

New language feature (J2SE 5.0) to iterate through arrays (or *collections*)

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

Equivalent to:

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

Summary for today

Programming Style

Loops

Arrays

Assignment 3

A group of friends participate in the Boston Marathon.

Find the best performer.

Find the second-best performer.