

Expressions and Control Statements

Announcements

- Programming Assignment #1 Out:
 - Karel the Robot: Due Friday, January 18 at 3:15 PM.
 - Email: Due Sunday, January 20 at 11:59PM.
 - Need help?
 - Stop by the LaIR!
 - Stop by our office hours!
 - Ask your section leader!
- Section assignments mailed out yesterday; sections start today.
- Ready to start coding in Java? Check out the **Blank Java Project** link on the CS106A website!
- Did you submit assignments before Tuesday? If so, can you please resubmit?

In the News

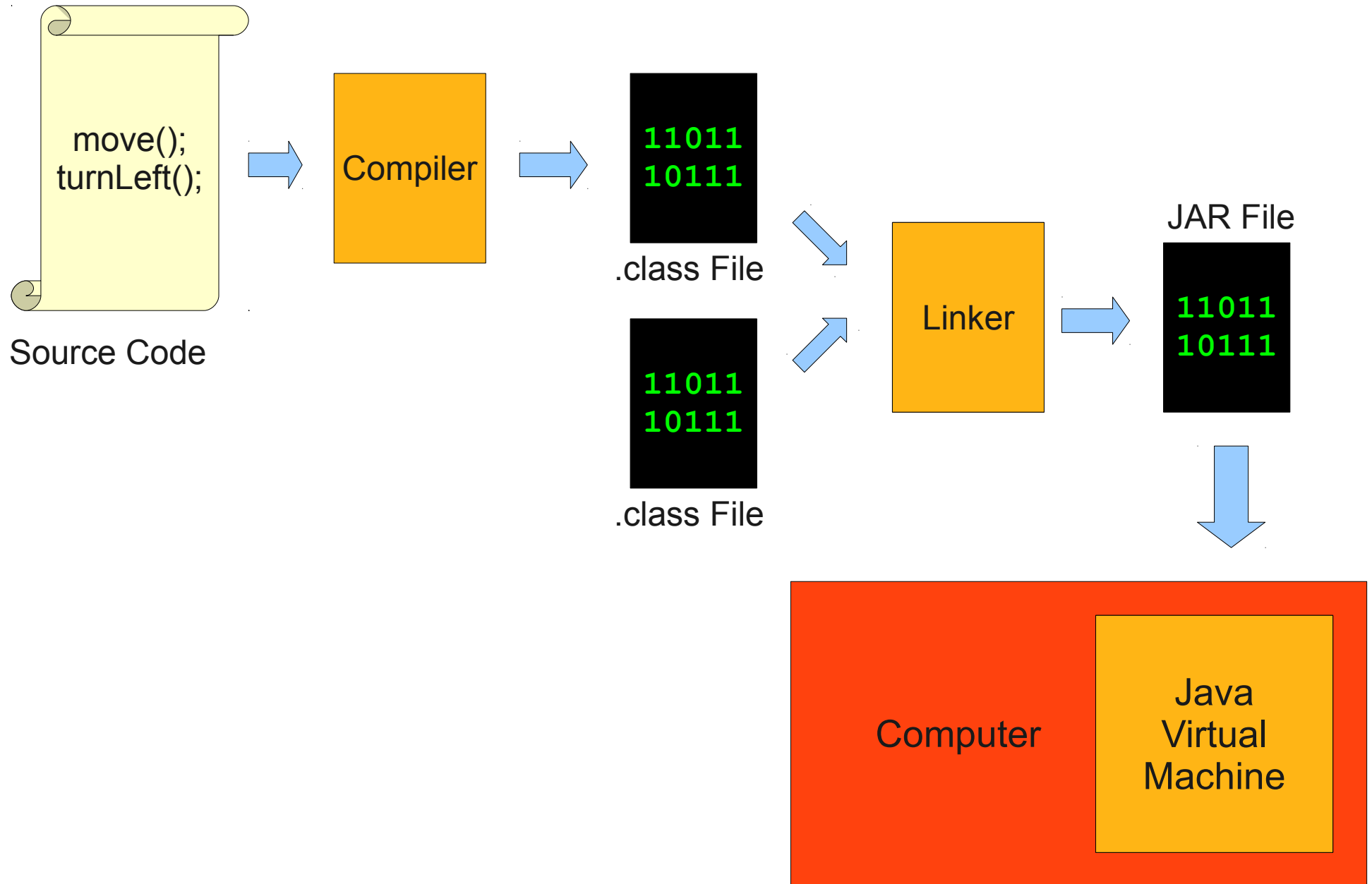


The screenshot shows the US-CERT website header with the Department of Homeland Security logo and the text "US-CERT UNITED STATES COMPUTER EMERGENCY READINESS TEAM". A navigation bar includes links for HOME, SECURITY PUBLICATIONS, ALERTS AND TIPS, RELATED RESOURCES, ABOUT US, and GFIRS. The main content area features a red heading "Alert (TA13-010A)" followed by the title "Oracle Java 7 Security Manager Bypass Vulnerability". Below the title, it states "Original release date: January 10, 2013 | Last revised: January 14, 2013". There are four social sharing buttons: Print, Tweet, Send, and Share. A section titled "Systems Affected" lists the following items:

- Java Platform Standard Edition 7 (Java SE 7)
- Java SE Development Kit (JDK 7)
- Java SE Runtime Environment (JRE 7)
- OpenJDK 7 and 7u
- IcedTea 2.x (IcedTea7 2.x)

At the bottom, a note states: "All versions of Java 7 through update 10 are affected. Web browsers using the Java 7 plug-in are at high risk."

The Java Model



Recap From Last Time

Variables

- A **variable** is a location where a program can store information for later use.

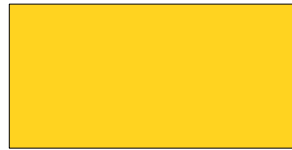
Variables

- A **variable** is a location where a program can store information for later use.



Variables

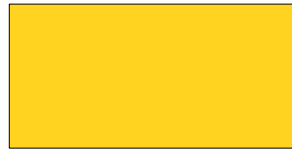
- A **variable** is a location where a program can store information for later use.



- Each variable has three pieces of information associated with it:

Variables

- A **variable** is a location where a program can store information for later use.



- Each variable has three pieces of information associated with it:
 - **Name**: What is the variable called?

Variables

- A **variable** is a location where a program can store information for later use.



`numVoters`

- Each variable has three pieces of information associated with it:
 - **Name**: What is the variable called?

Variables

- A **variable** is a location where a program can store information for later use.



`numVoters`

- Each variable has three pieces of information associated with it:
 - **Name**: What is the variable called?
 - **Type**: What sorts of things can you store in the variable?

Variables

- A **variable** is a location where a program can store information for later use.

 `int numVoters`

- Each variable has three pieces of information associated with it:
 - **Name**: What is the variable called?
 - **Type**: What sorts of things can you store in the variable?

Variables

- A **variable** is a location where a program can store information for later use.

 `int numVoters`

- Each variable has three pieces of information associated with it:
 - **Name**: What is the variable called?
 - **Type**: What sorts of things can you store in the variable?
 - **Value**: What value does the variable have at any particular moment in time?

Variables

- A **variable** is a location where a program can store information for later use.

137 int numVoters

- Each variable has three pieces of information associated with it:
 - **Name**: What is the variable called?
 - **Type**: What sorts of things can you store in the variable?
 - **Value**: What value does the variable have at any particular moment in time?

Expressions

Expressions

- Variables and other values can be used in **expressions**.
- Some familiar mathematical operators:
 - + (addition)
 - - (subtraction)
 - * (multiplication)
 - / (division)

The Remainder Operator

- The special operator `%` computes the **remainder** of one value divided by another.
- $a \% b$ is pronounced “ a mod b .”
- For example:
 - $15 \% 3 = 0$
 - $14 \% 8 = 6$
 - $21 \% 2 = 1$
 - $14 \% 17 = 14$

Operator Precedence

- Java's mathematical operators have the following precedence:
 - $()$ (*highest*)
 - $*$ / $\%$
 - $+$ - (*lowest*)
- Operators of equal precedence are evaluated left-to-right.

Fun with Division



she got more
than me!





Cookies for everyone!

A Useful Shorthand

- Commonly, programs contain code like this:

```
x = x + 1;
```

```
z = z / 14;
```

```
y = y * 137;
```

```
w = w - 3;
```

A Useful Shorthand

- Commonly, programs contain code like this:

```
x = x + 1;
```

```
y = y * 137;
```

```
z = z / 14;
```

```
w = w - 3;
```

- The statement

variable = variable op value ;

can be rewritten as

variable op= value ;

A Useful Shorthand

- Commonly, programs contain code like this:

```
x += 1;
```

```
y *= 137;
```

```
z /= 14;
```

```
w -= 3;
```

- The statement

variable = variable op value ;

can be rewritten as

variable op= value ;

Another Useful Shorthand

- In the special case of writing

variable = ***variable*** + 1 ;

we can instead write

variable ++ ;

- In the special case of writing

variable = ***variable*** - 1 ;

we can instead write

variable -- ;

Control Statements Revisited

Control Statements

`for`

`if`

`while`

Control Statements

for

if

while

This is called the **initialization statement** and is performed before the loop starts.

This is called the **step** or **increment** and is performed at the end of each loop iteration.

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

This is called the **loop condition** or **termination condition**. The loop will check whether this statement is true before each execution.

Video: Gangnam Style

Lyrics for International Superstardom

Oppan Gangnam Style
Gangnam Style

Op
Op
Op
Op

Oppan Gangnam Style
Gangnam Style

Op
Op
Op
Op

Oppan Gangnam Style

Lyrics for International Superstardom

Oppan Gangnam Style
Gangnam Style

Op
Op
Op
Op

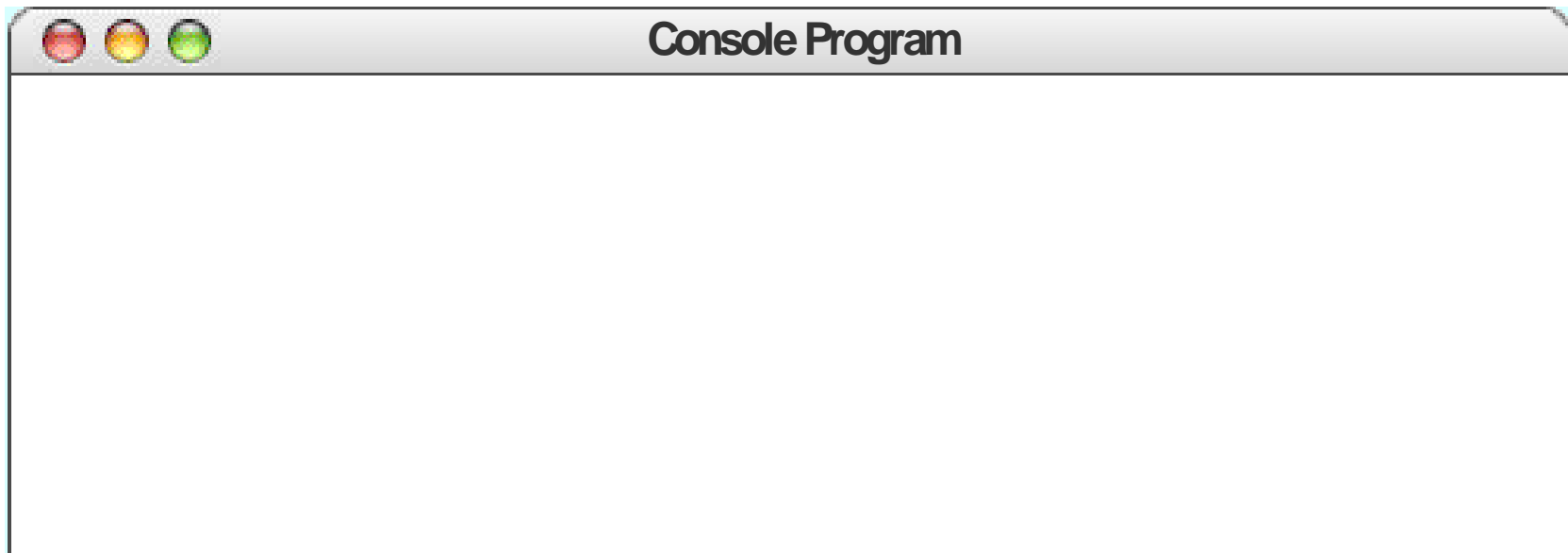
Oppan Gangnam Style
Gangnam Style

Op
Op
Op
Op

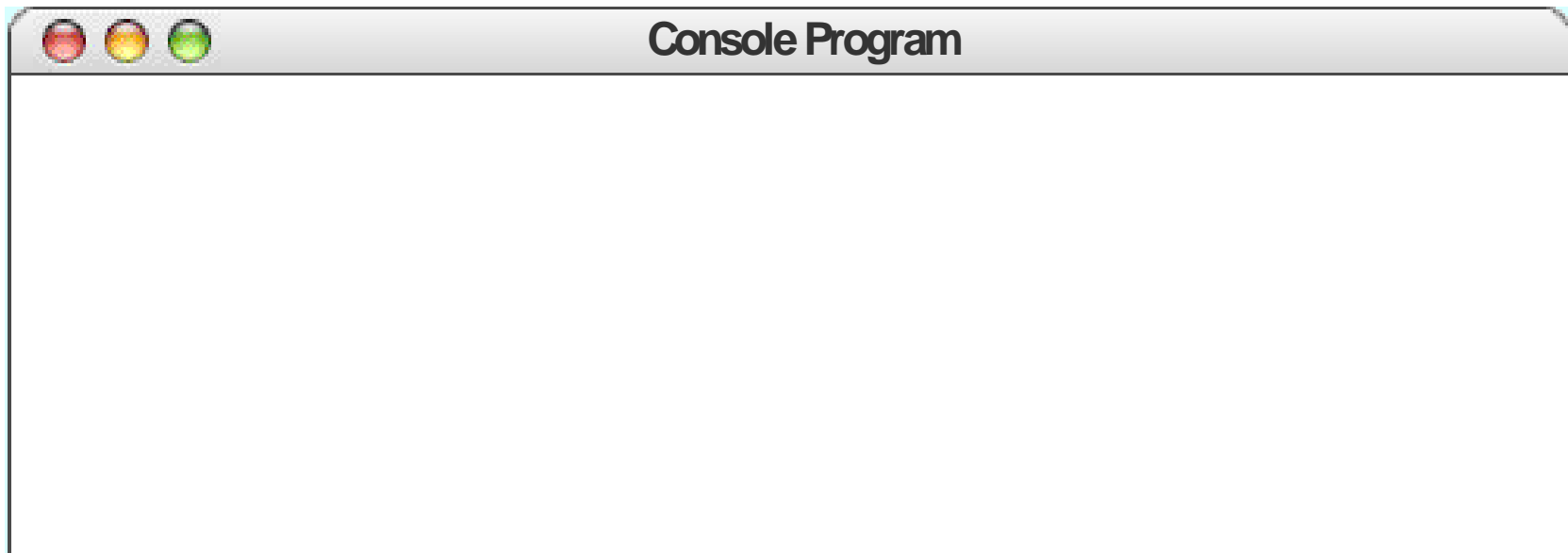
Oppan Gangnam Style

```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

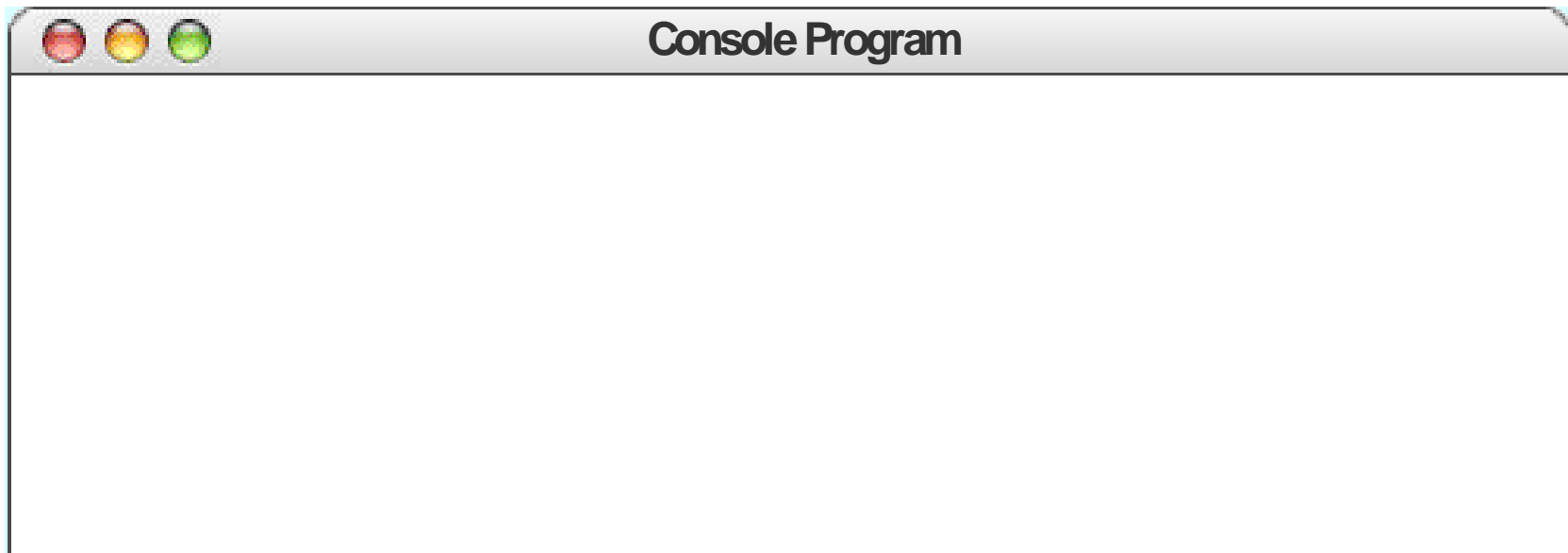


```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```



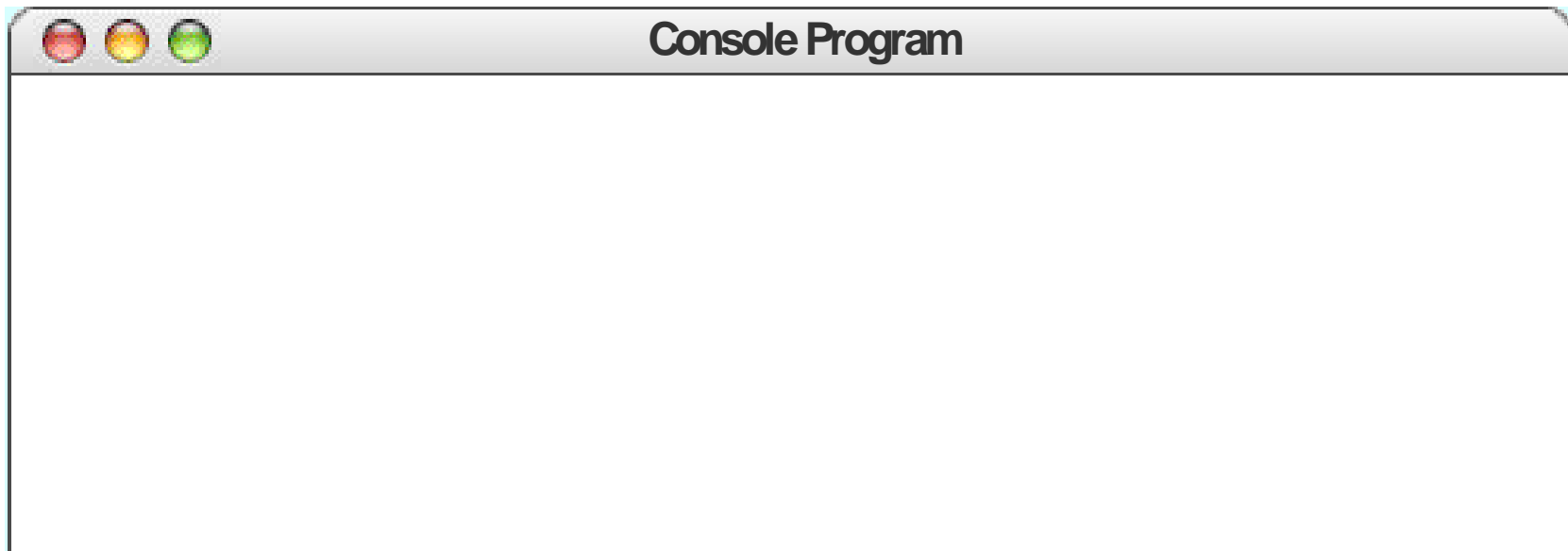
```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

int i 0



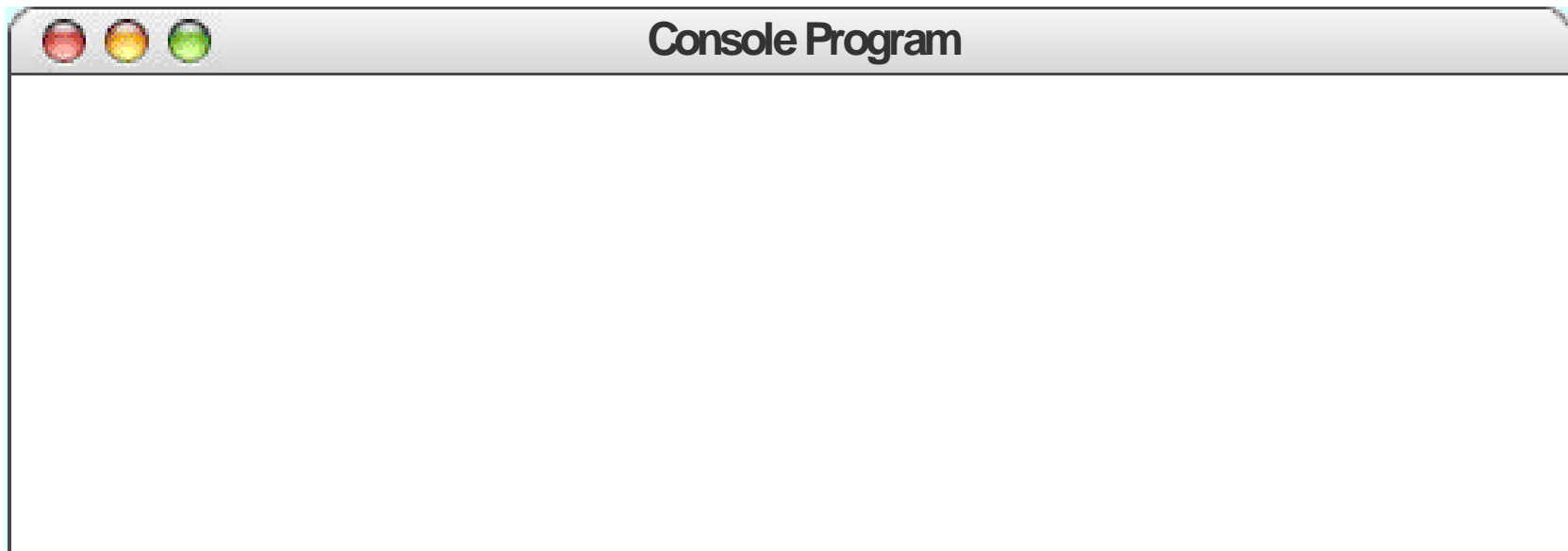
```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

int i 0



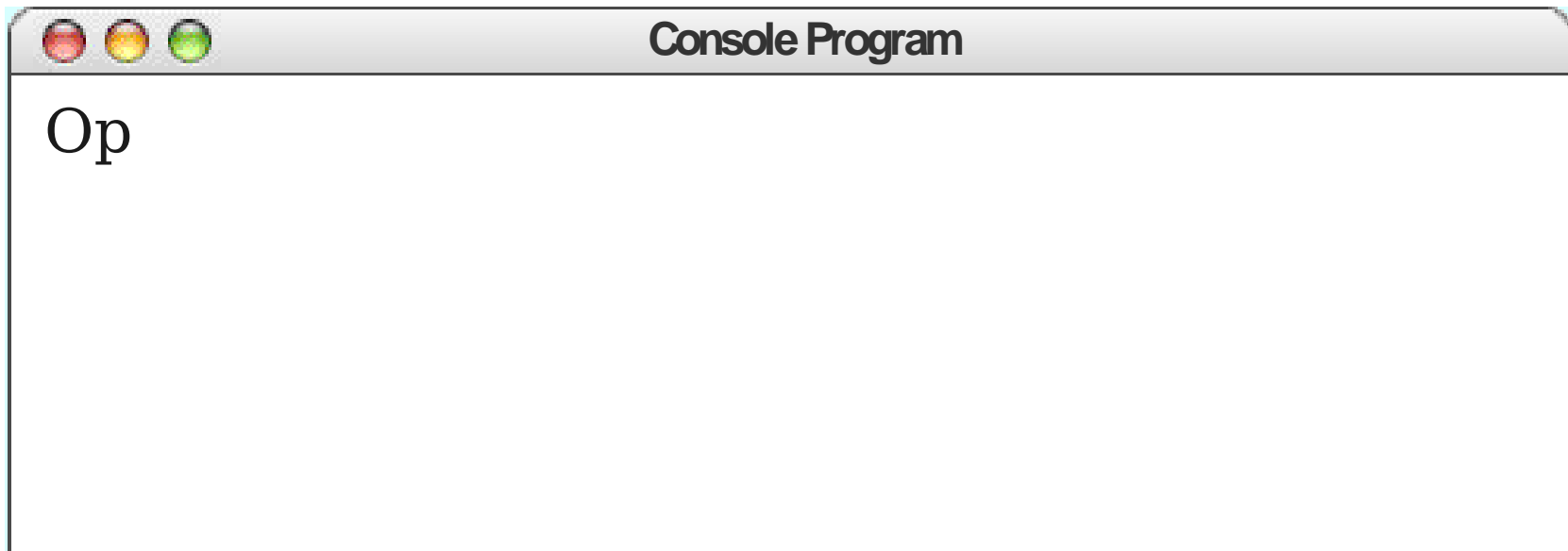
```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

```
int i
```



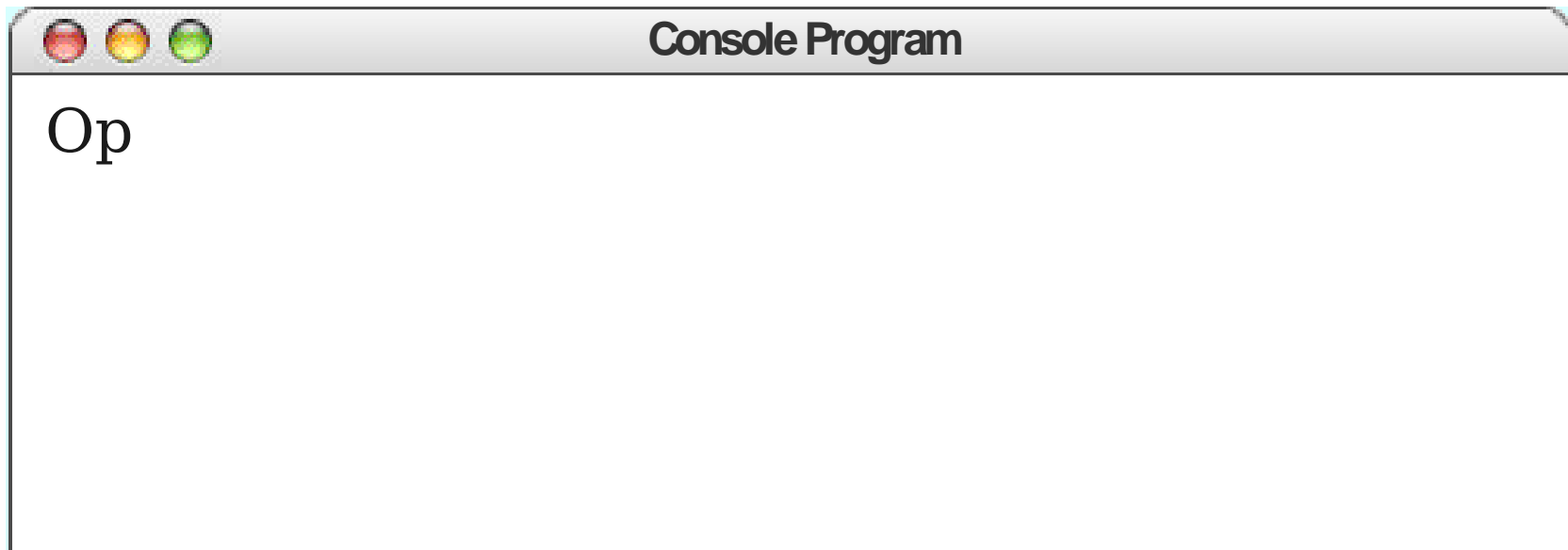
```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

int i 




```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

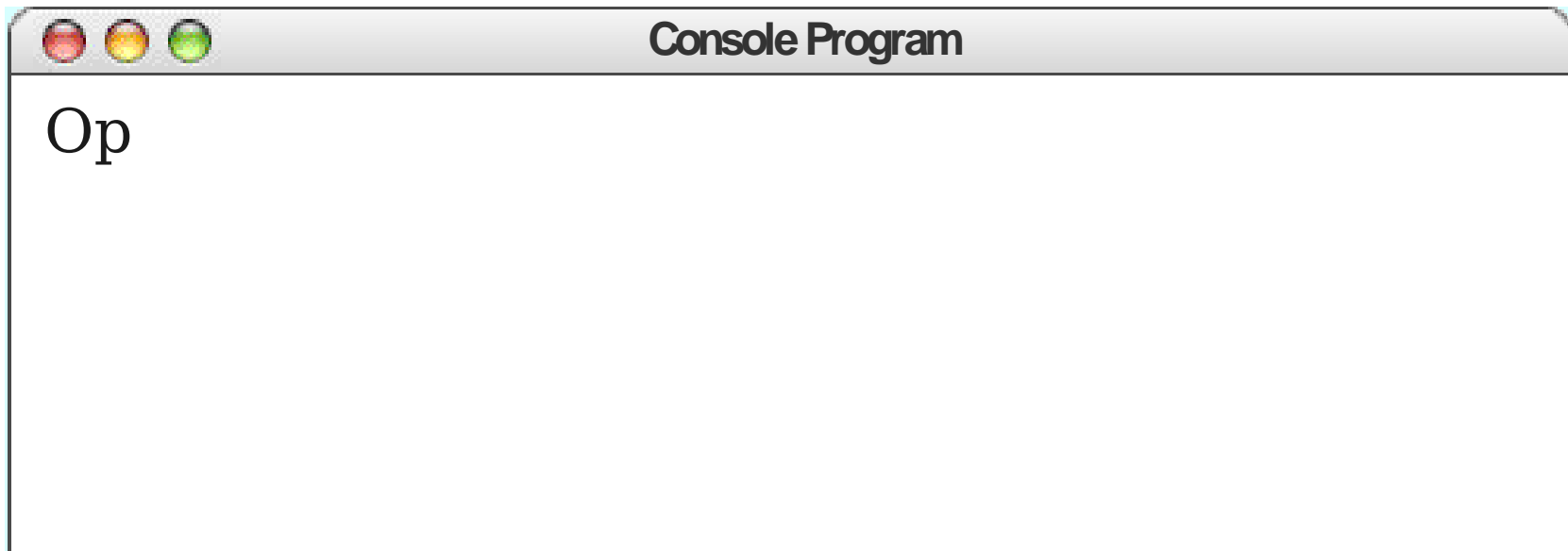
int i 0



```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

```
int i
```

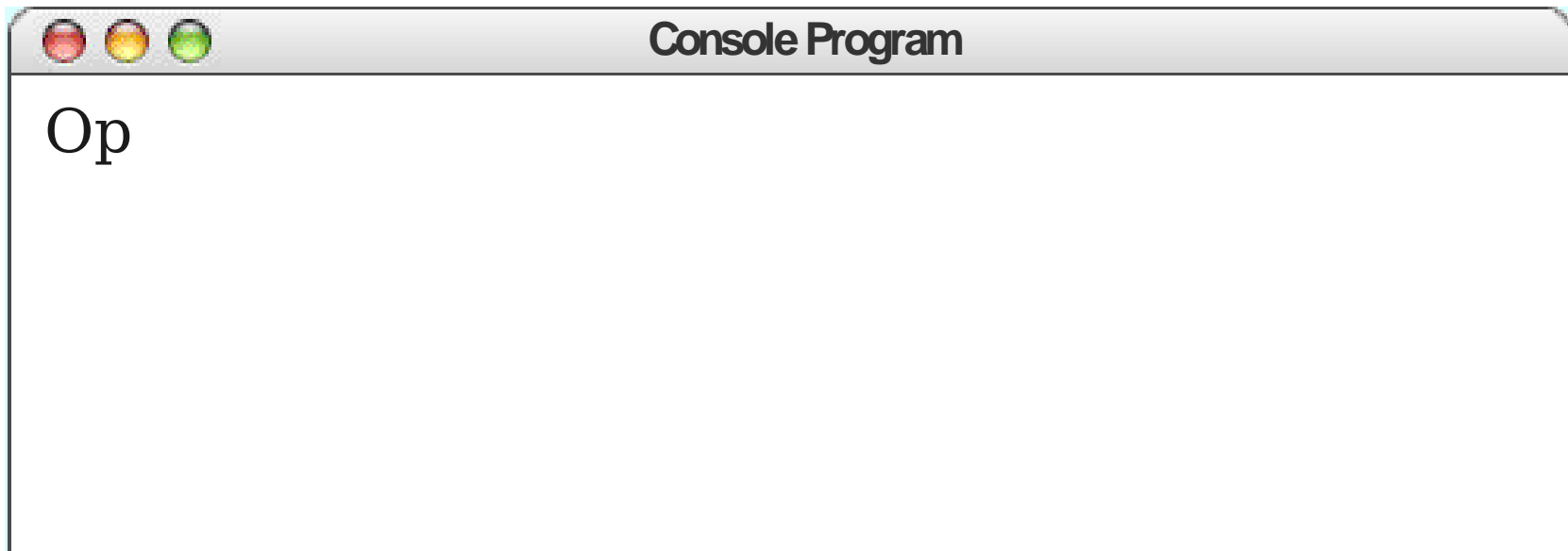
1



```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

```
int i
```

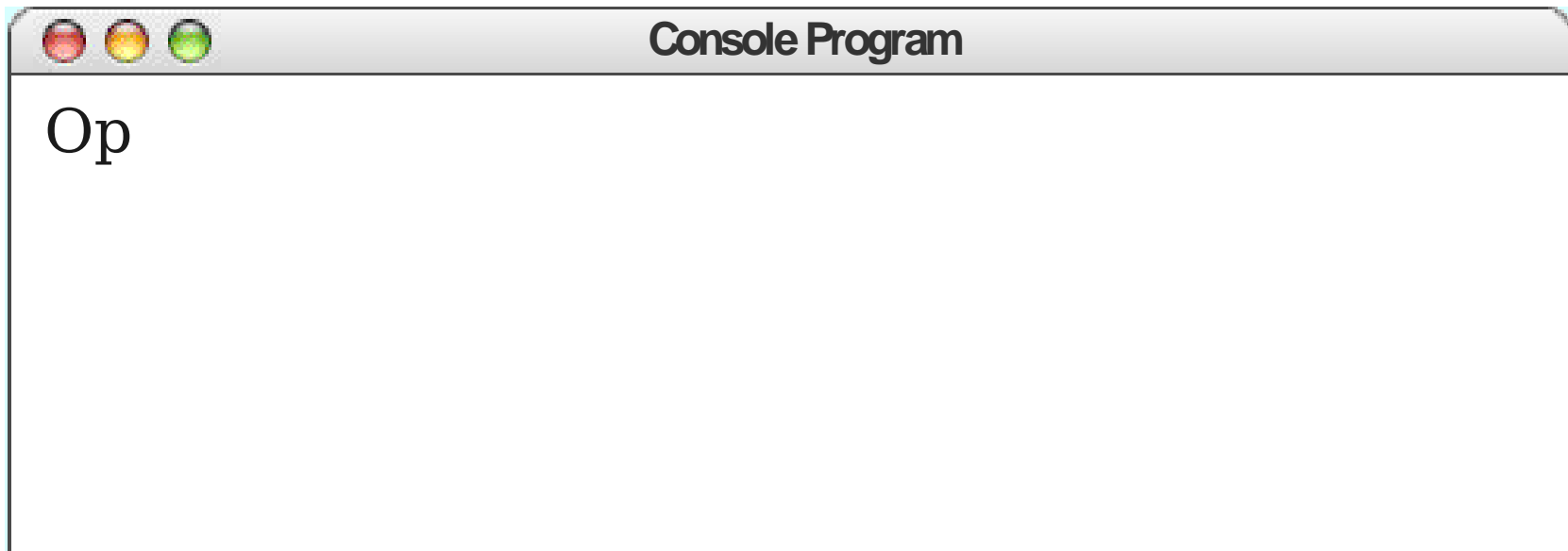
1



```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

```
int i
```

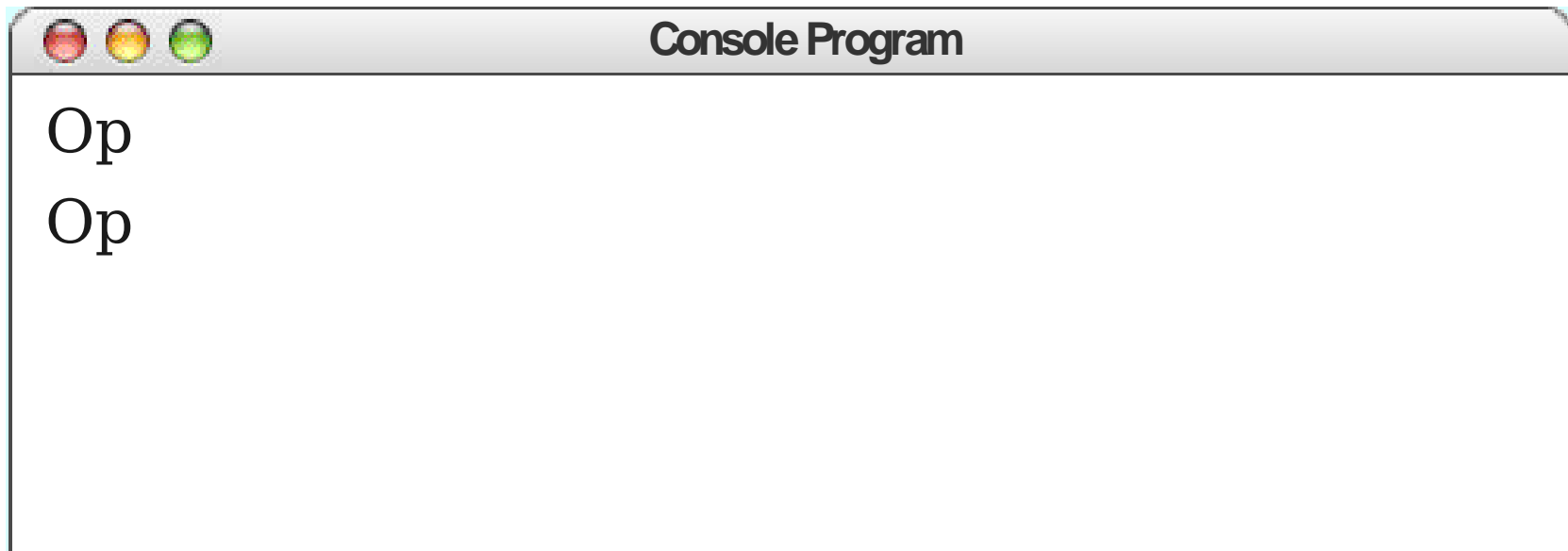
1



```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

```
int i
```

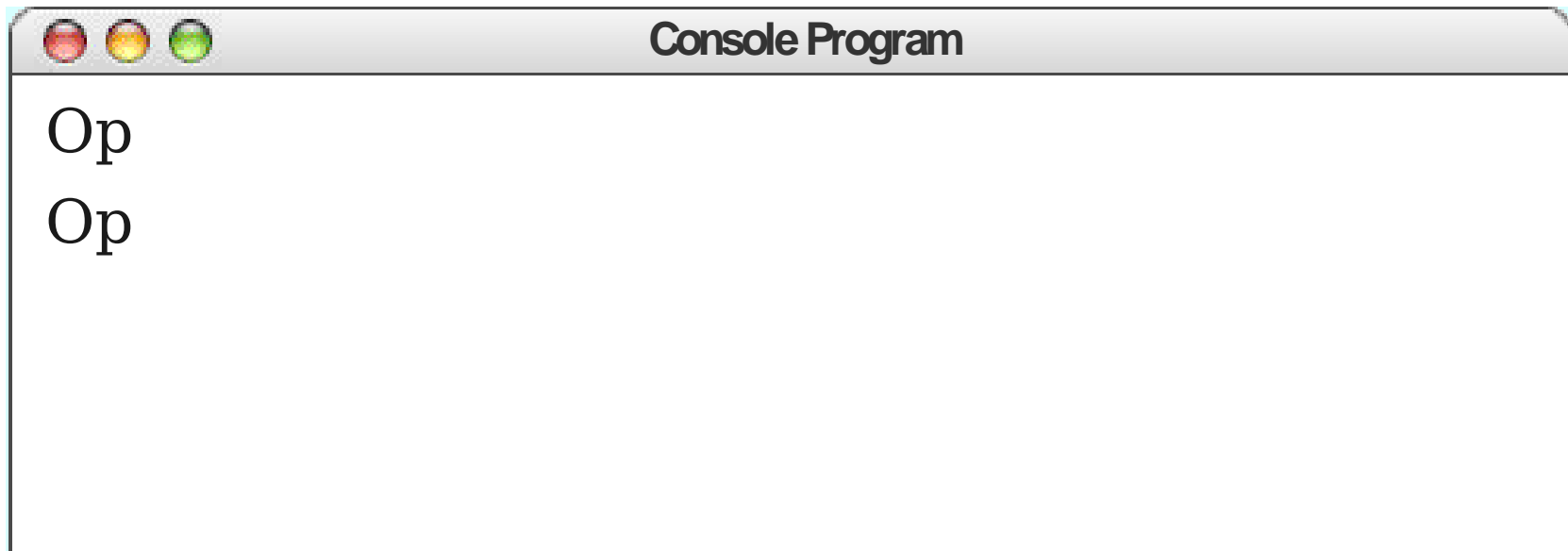
1



```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

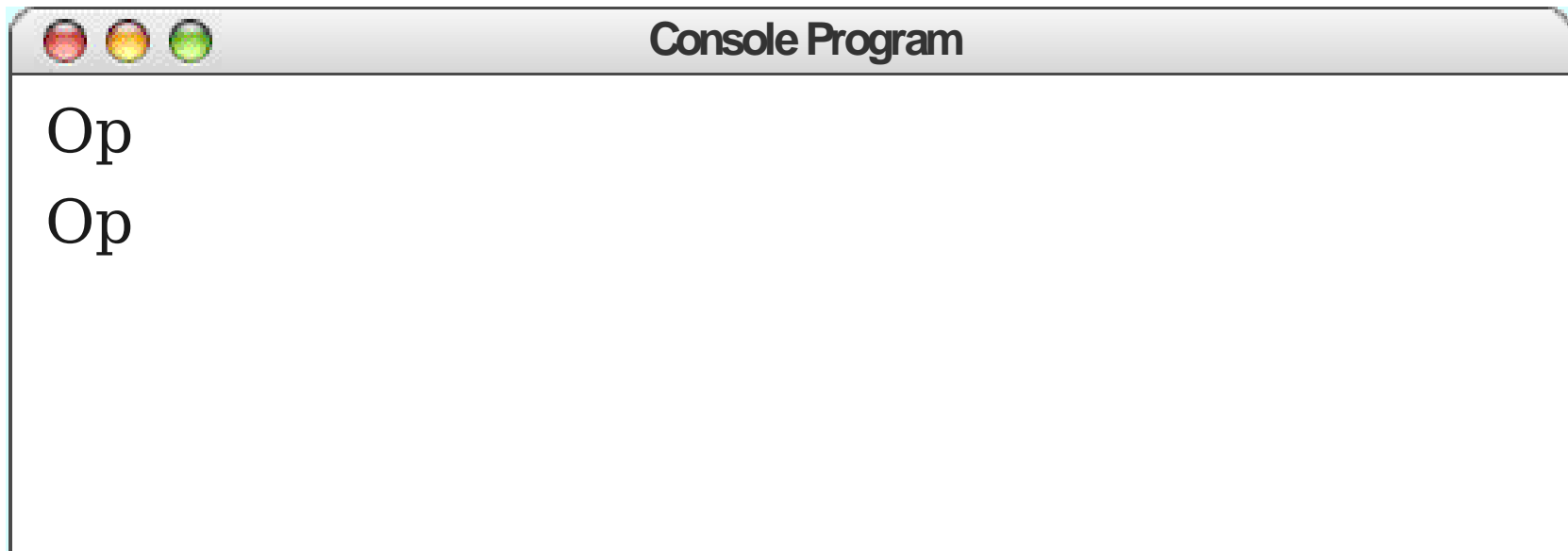
```
int i
```

1



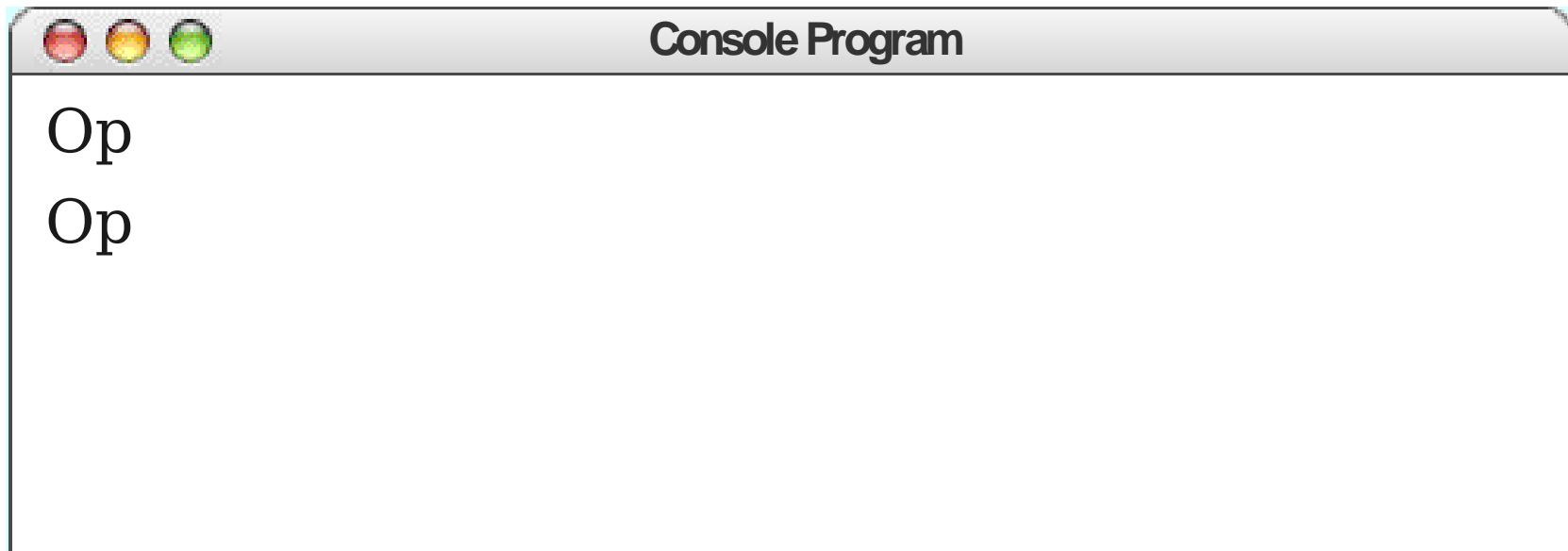
```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

int i 2



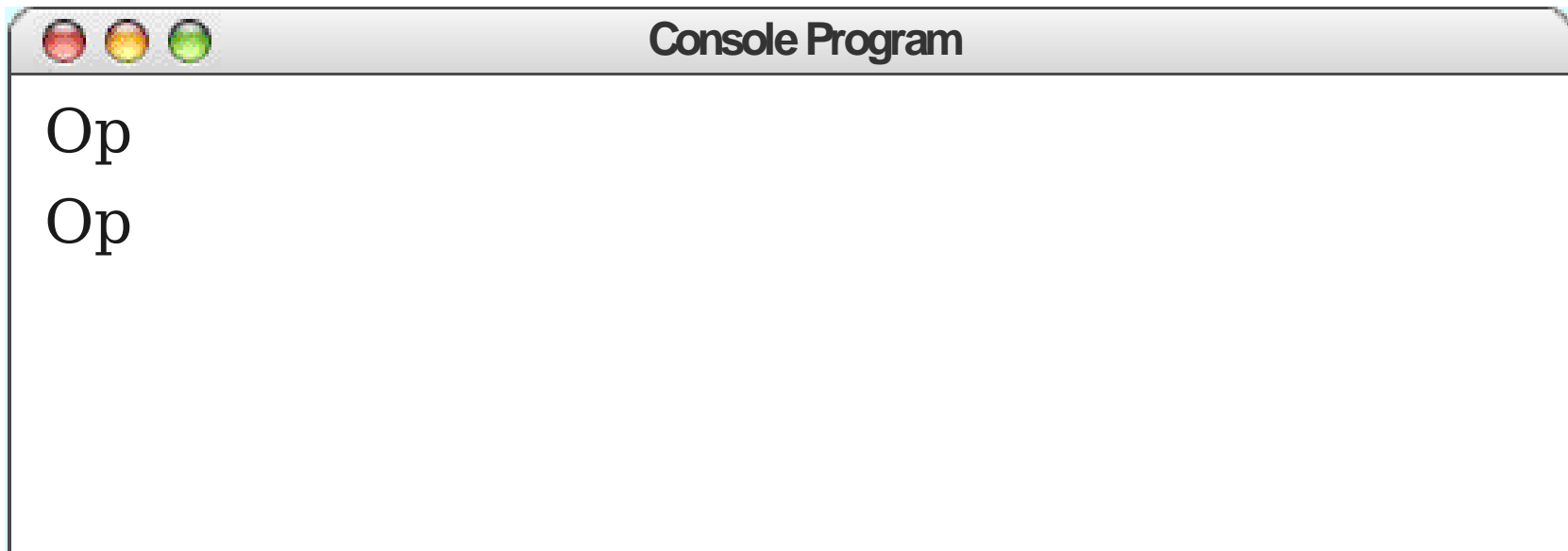
```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

int i 2



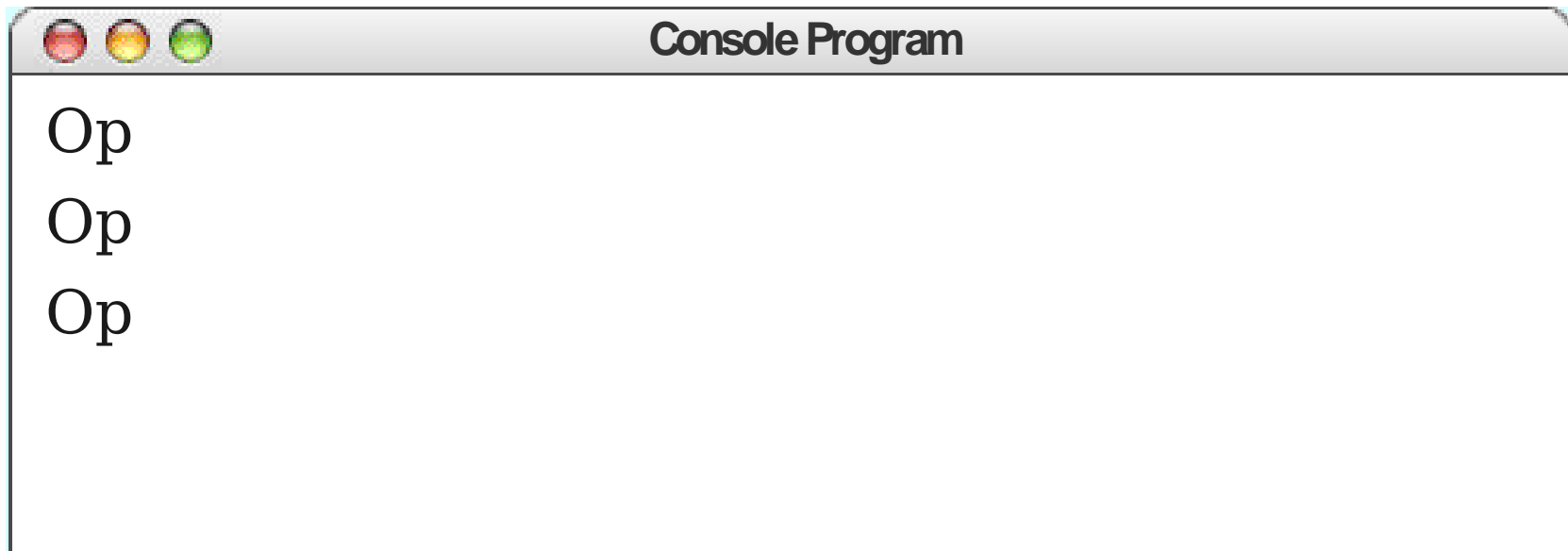

```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

```
int i
```



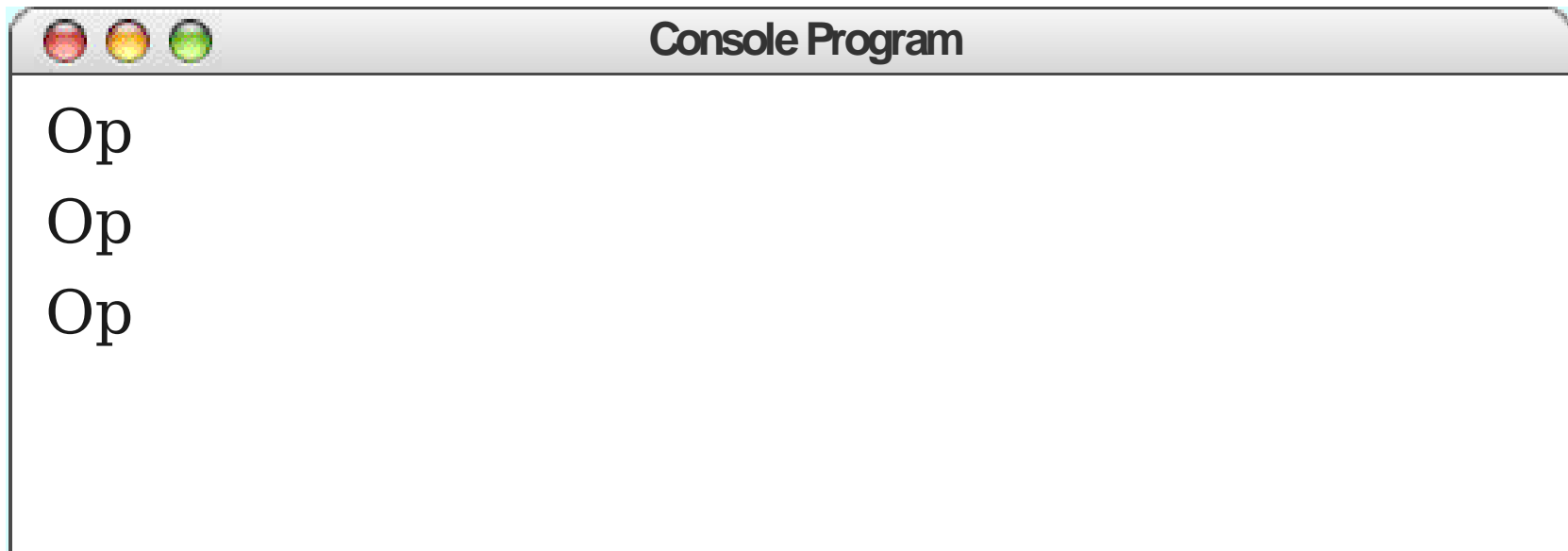
```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

int i 2



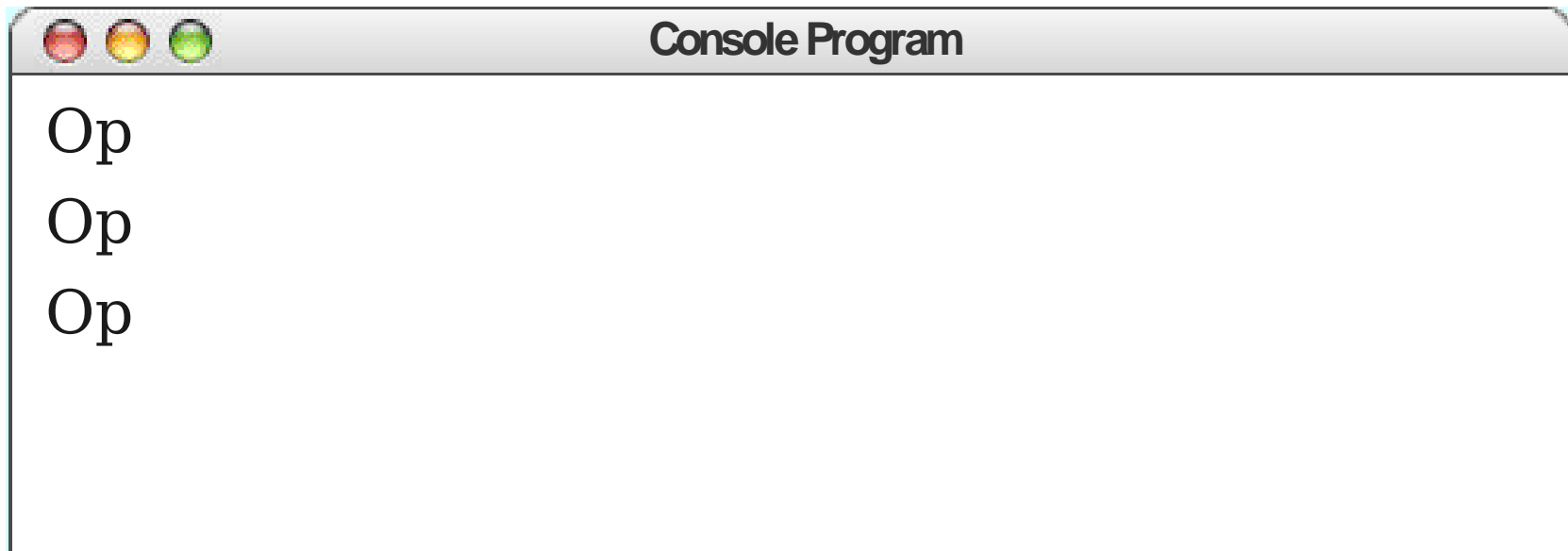
```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

int i 2



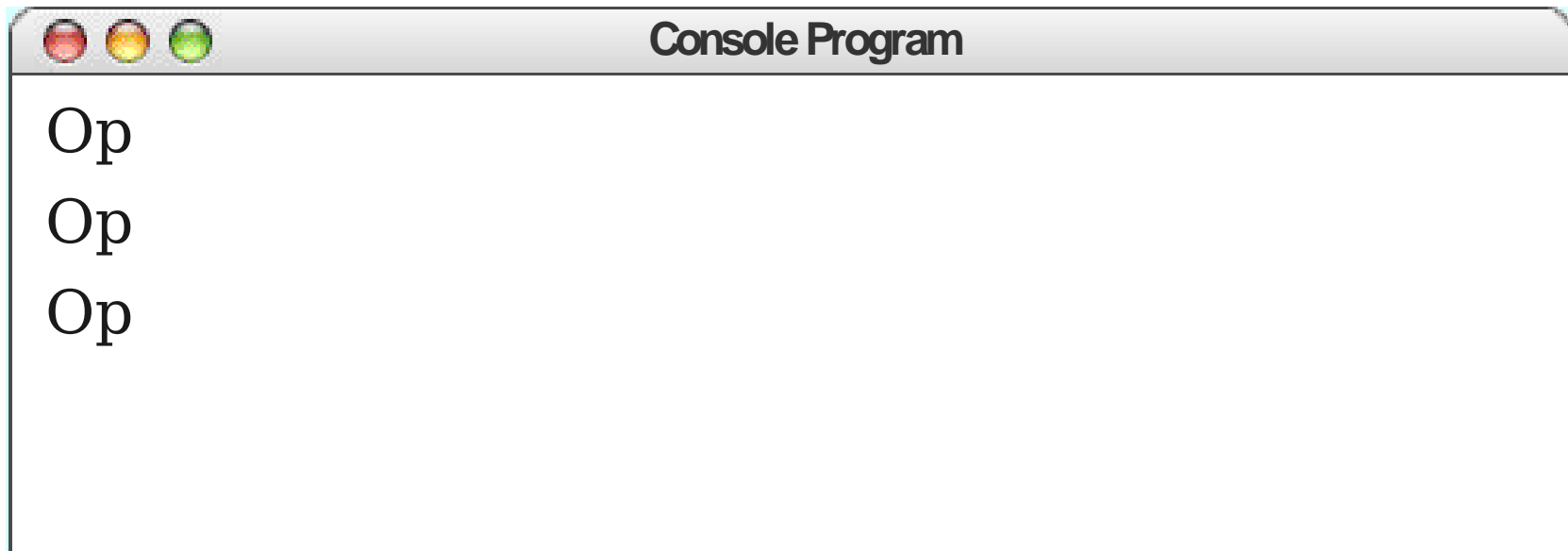
```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

int i 3



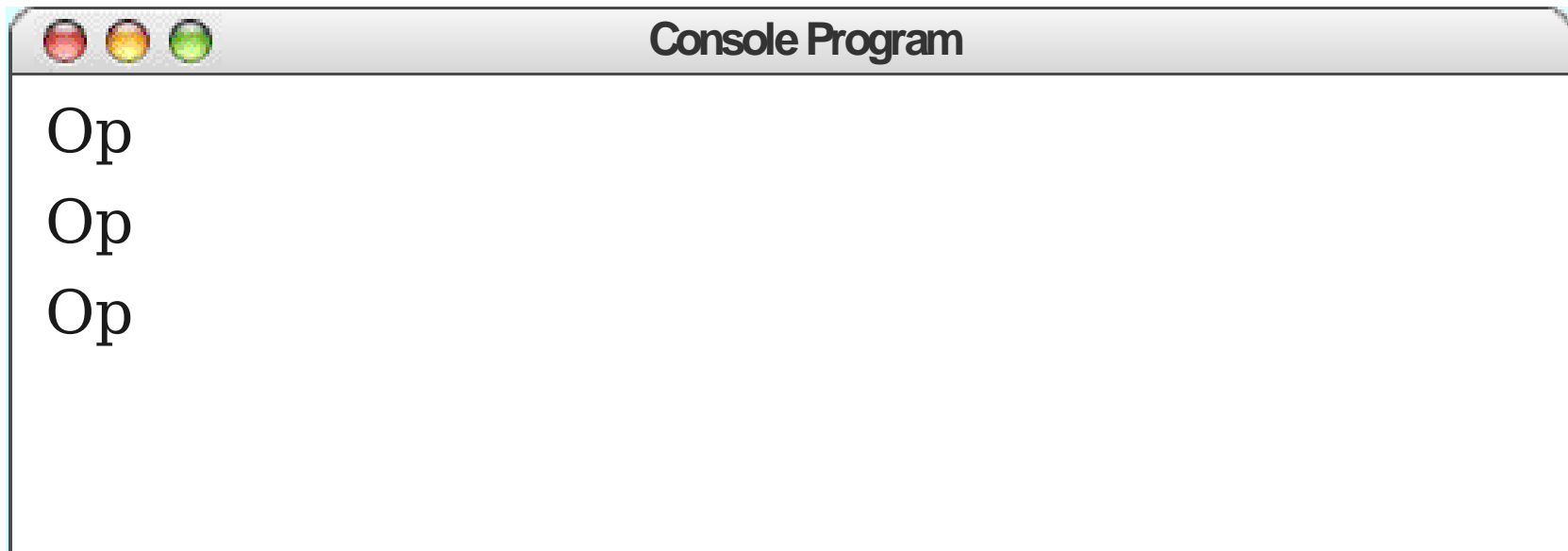
```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

int i 3



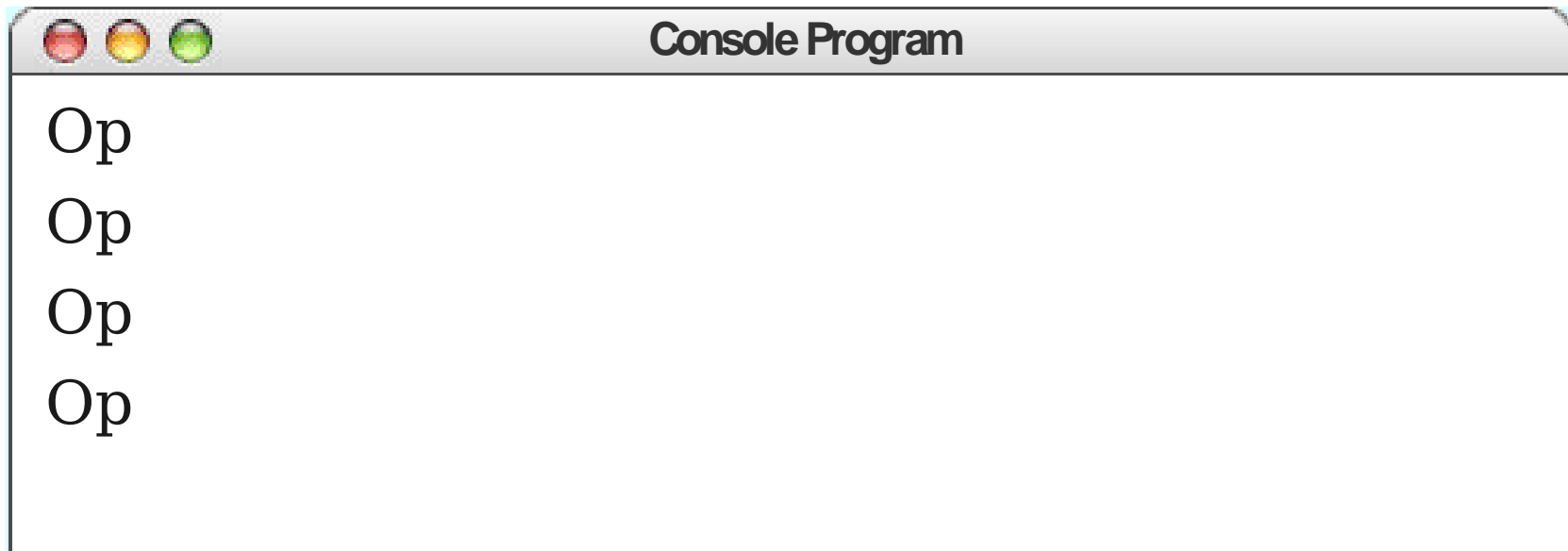
```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

int i 3



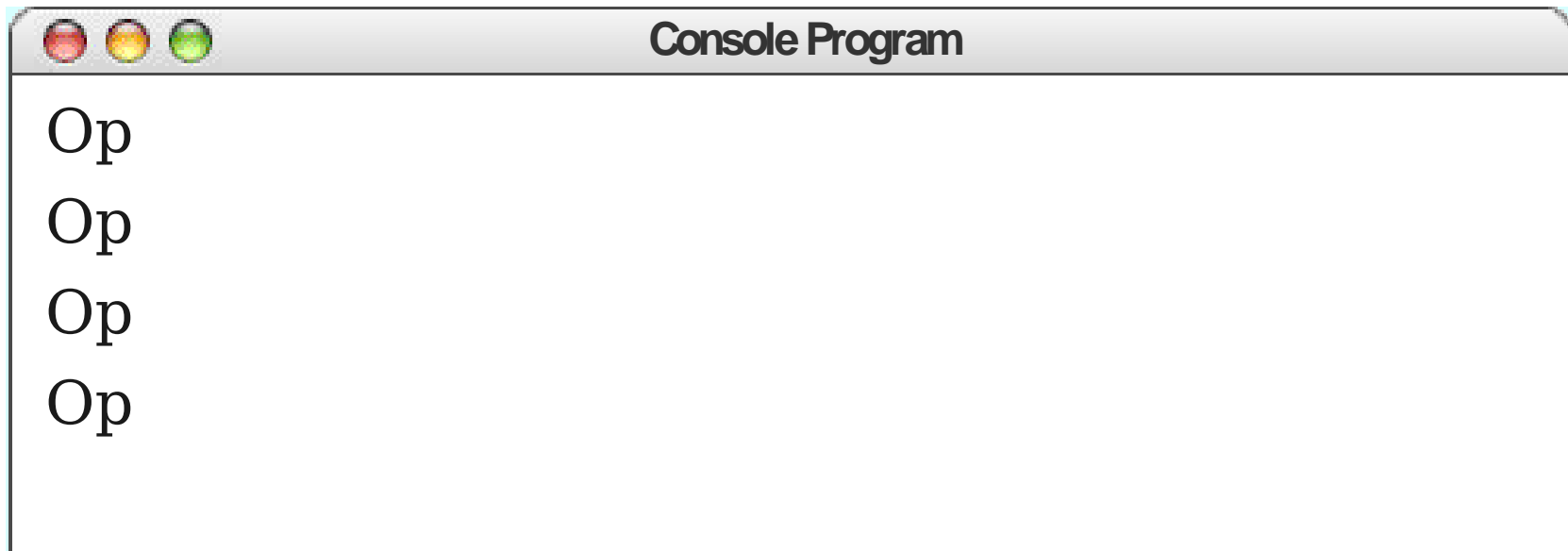
```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

int i 3



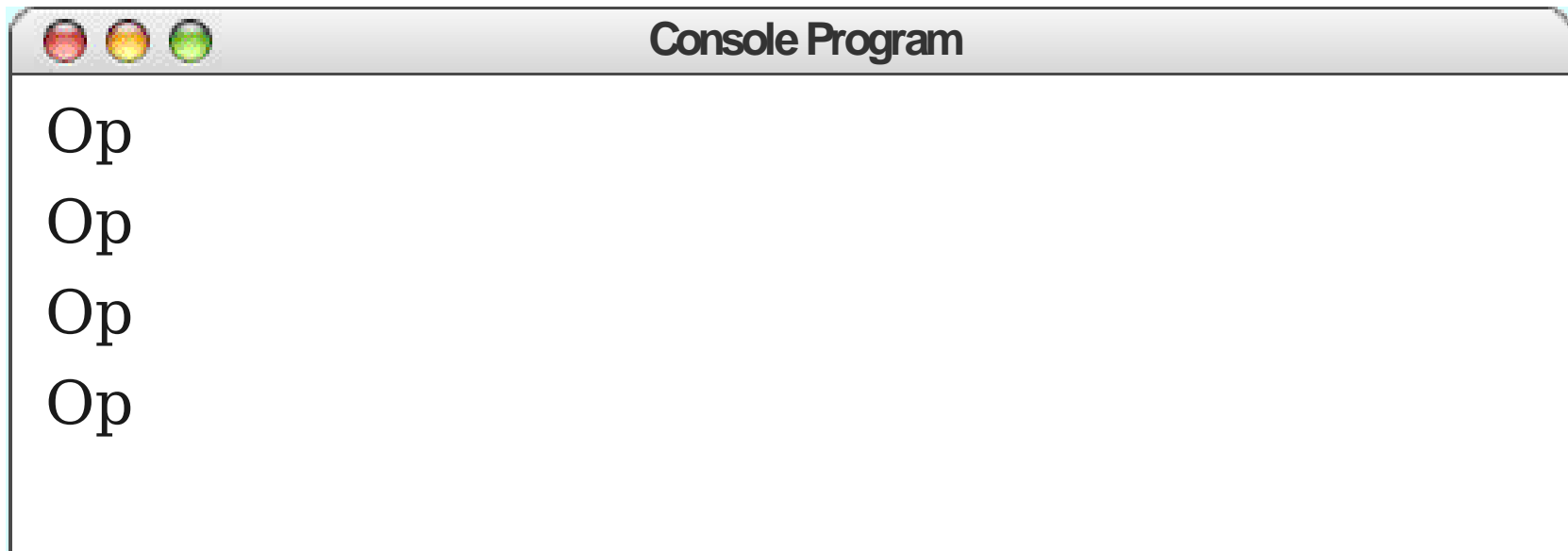
```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

int i 3



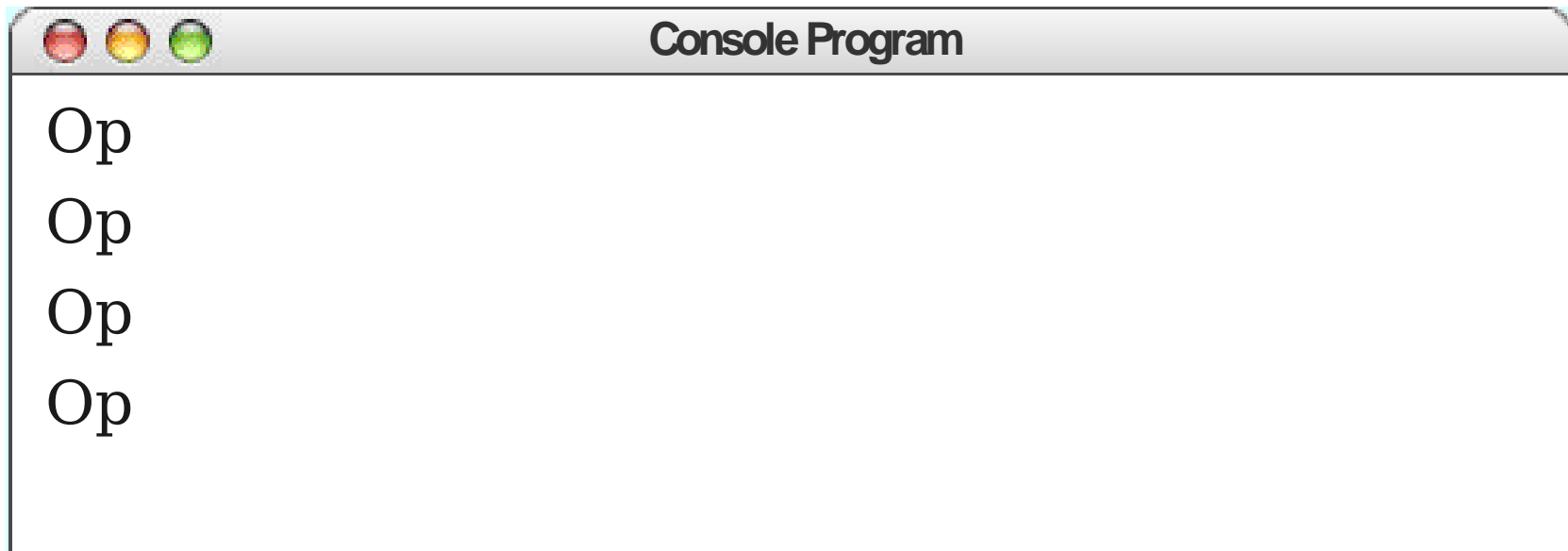

```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

```
int i
```



```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

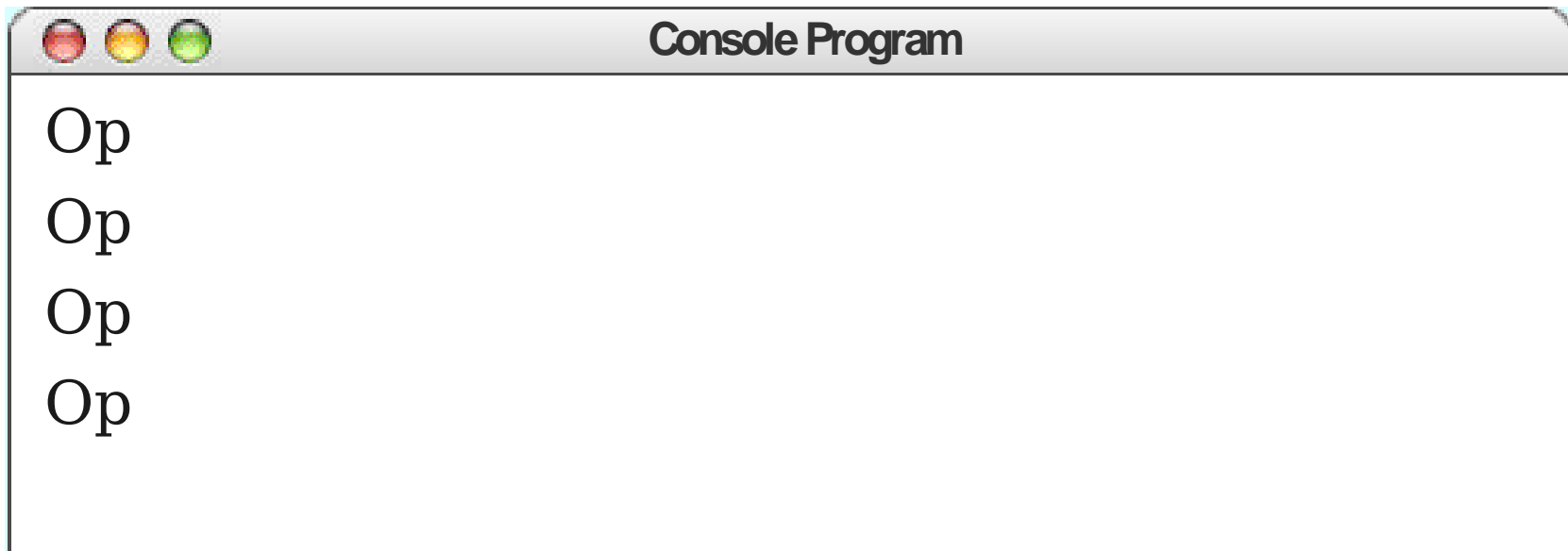
int i 4



```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}
```

```
println("Oppan Gangnam Style");
```

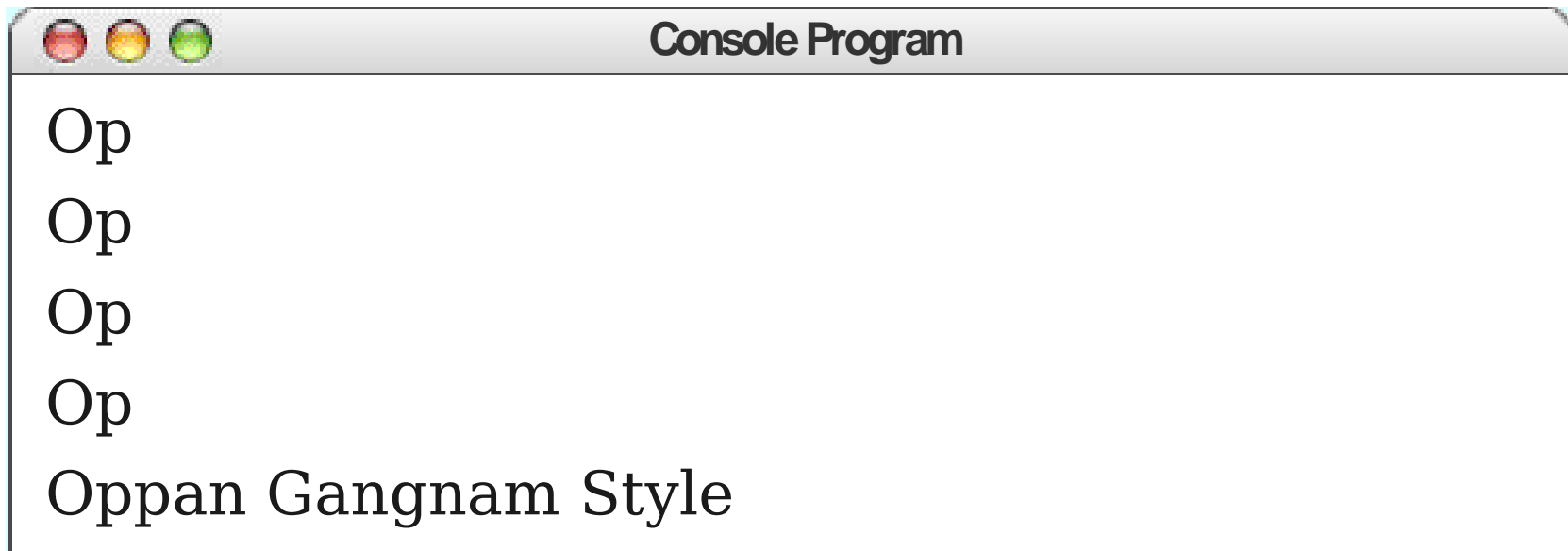
```
int i
```



```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}
```

```
println("Oppan Gangnam Style");
```

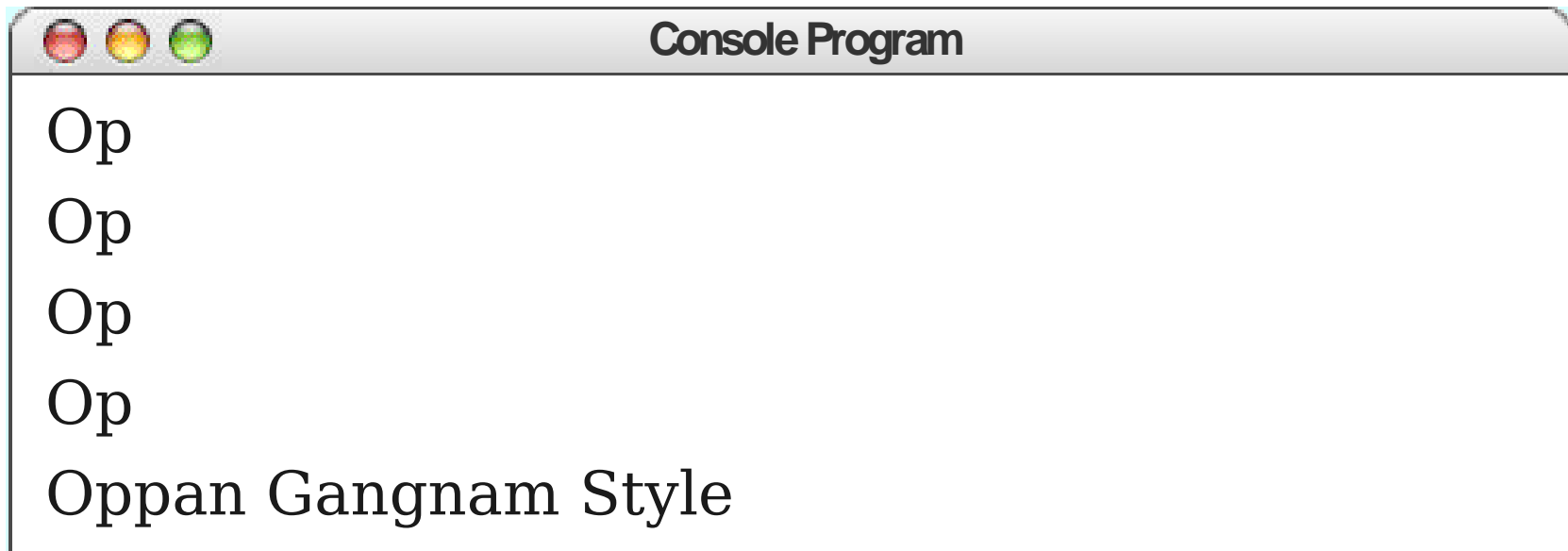
```
int i
```



```
Console Program  
Op  
Op  
Op  
Op  
Oppan Gangnam Style
```

```
for (int i = 0; i < 4; i++) {  
    println("Op");  
}  
println("Oppan Gangnam Style");
```

int i 4



Console Program

```
Op  
Op  
Op  
Op  
Oppan Gangnam Style
```

Oppan Gangnam Style
Gangnam Style

Op

Op

Op

Op

Oppan Gangnam Style
Gangnam Style

Op

Op

Op

Op

Oppan Gangnam Style

Oppan Gangnam Style
Gangnam Style

Op

Op

Op

Op

Oppan Gangnam Style
Gangnam Style

Op

Op

Op

Op

Oppan Gangnam Style

Boolean Expressions

- A **boolean expression** is a test for a condition (it is either **true** or **false**).
- Value comparisons:
 - `==` “equals” *(note: not single =)*
 - `!=` “not equals” *(cannot say <>)*
 - `>` “greater than”
 - `<` “less than”
 - `>=` “greater than or equal to”
 - `<=` “less than or equal to”

Logical Operators

- We can apply **logical operators** to boolean values to produce new values.
- Logical **NOT**: `!p`
 - `!p` is **true** if `p` is **false**; `!p` is **false** if `p` is **true**.
- Logical **AND**: `p && q`
 - `p && q` is **true** when both `p` and `q` are true.
- Logical **OR**: `p || q`
 - `p || q` is **true** when `p` is true, `q` is true, or both `p` and `q` are true.
- Order of precedence given above.

Short-Circuit Evaluation

- Cute observations:
 - `true || p` is always `true`.
 - `false && p` is always `false`.
- The logical operators **short-circuit**: if the answer is known from the left operand, the right side is not computed.
- Example: The code
`boolean b = (x == 0) || ((y / x) < 20)`
will never divide by zero.