

# Primitive Types

Integers



# Lecture Contents

- Review of Number Systems
- How to Store a Binary Integer in a Computer
- Java `int` type



# Review of Number Systems

- The **denary** (a.k.a. **decimal**) system uses **place value**, ten digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), and a decimal point to represent a number.





# Review of Number Systems

- The **binary** system uses **place value** and two digits (0, 1) to represent a value.
- Although we could, we generally don't see a decimal point used with binary numbers.
- For denary, each place value is a power of ten. For binary, each place value is a power of two.

$10^4$	$10^3$	$10^2$	$10^1$	$10^0$

$128$ $2^7$	$64$ $2^6$	$32$ $2^5$	$16$ $2^4$	$8$ $2^3$	$4$ $2^2$	$2$ $2^1$	$1$ $2^0$



# Review of Number Systems

- Be able to convert back and forth between *binary* and *denary*.

$$75 - 64 = 11$$

$$11 - 8 = 3$$

$$3 - 2 = 1$$

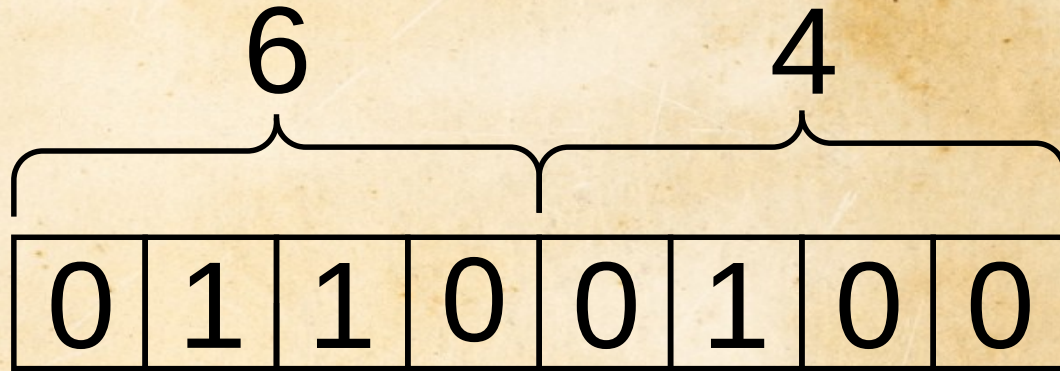
$$1 - 1 = 0$$

$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
0	1	0	0	1	0	1	1
128	64	32	16	8	4	2	1



# Review of Number Systems

- Be able to convert back and forth between *binary* and *hexadecimal*.



Decimal	Hexa- decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111



# Lecture Contents

- Review of Number Systems
- **How to Store a Binary Integer in a Computer**
- Java `int` type



# How to Store a Binary Integer in a Computer

- Modern computers organize **bits** into **bytes** (8 bits = 1 byte).
- Computer hardware is designed to operate on a set number of bytes.
  - The more bytes we operate on at once, the more complex the hardware must be.
- How many bytes should we use to store each integer?



# How to Store a Binary Integer in a Computer

- Modern computers organize **bits** into **bytes** (8 bits = 1 byte).
- Computer hardware is designed to operate on a set number of bytes
- How many bytes should we use to store each integer?

– 8 bits  $\rightarrow 2^8 = 256$

– 16 bits  $\rightarrow 2^{16} = 65536$

– 32 bits  $\rightarrow 2^{32} = 4,294,967,296$





# How to Store a Binary Integer in a Computer

- Modern computers organize **bits** into **bytes** (8 bits = 1 byte).
- Computer hardware is designed to operate on a set number of bytes
- How many bytes should we use to store each integer?

– 8 bits  $\rightarrow 2^8 = 256$

– 16 bits  $\rightarrow 2^{16} = 65536$

– 32 bits  $\rightarrow 2^{32} = 4,294,967,296$





# How to Store a Binary Integer in a Computer

- Java programmers can choose from the following *types* for storing integers

type	bits	numerical range
byte	8	-128 to +127
short	16	-32768 to +32767
int	32	-2,147,483,648 to +2,147,483,647
long	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

- For this course, unless you have a specific reason, always use the type **int** to store integers.



# Lecture Contents

- Review of Number Systems
- How to Store a Binary Integer in a Computer
- **Java `int` type**

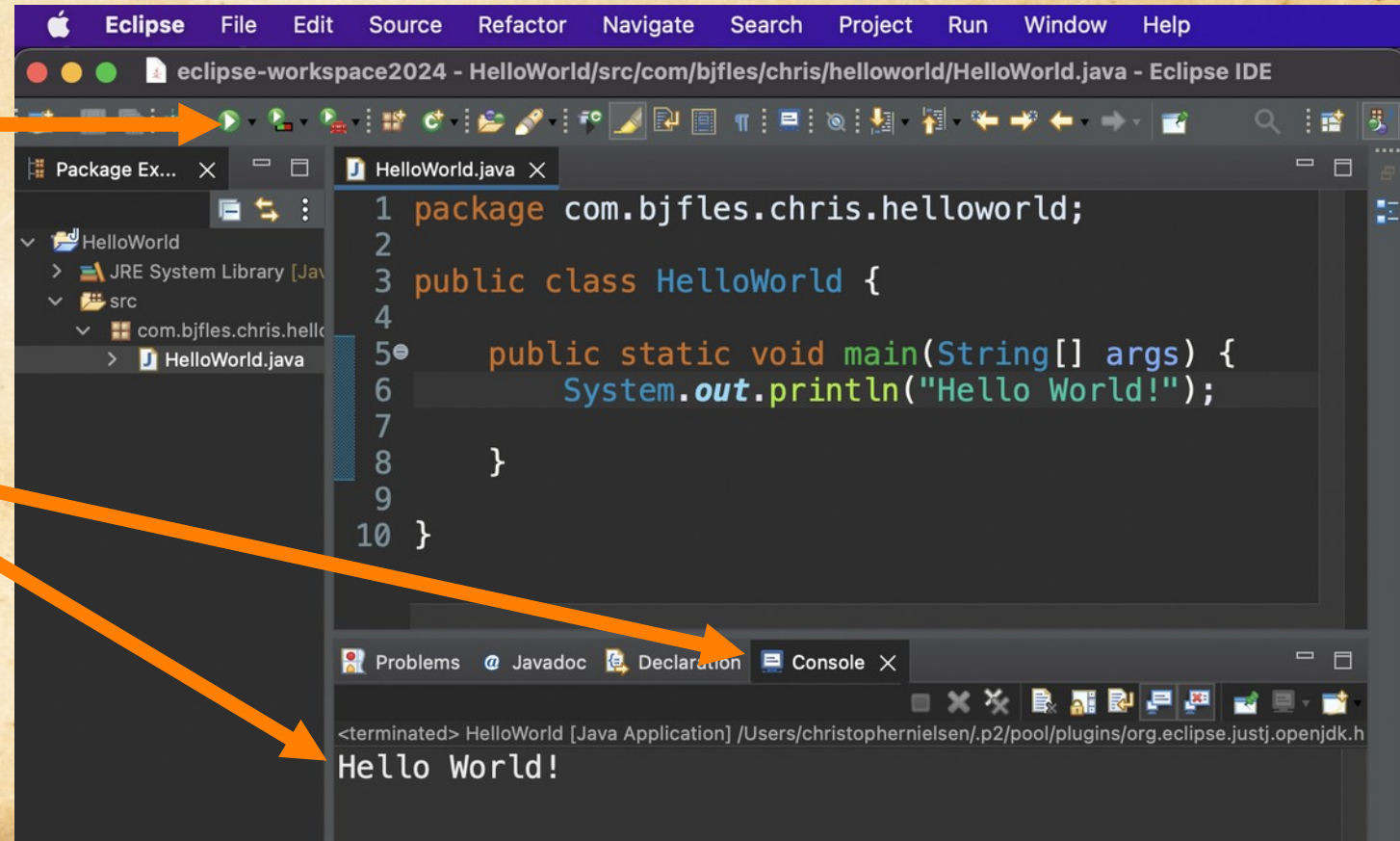


# Java **int** Type

Ensure you were able to get the “HelloWorld” assignment completed.

Click here  
to run.

Check the  
output in the  
console.



The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The title bar indicates the workspace is 'eclipse-workspace2024' and the current file is 'HelloWorld.java'. The left sidebar shows the project structure with 'HelloWorld' and its sub-packages. The main editor displays the following Java code:

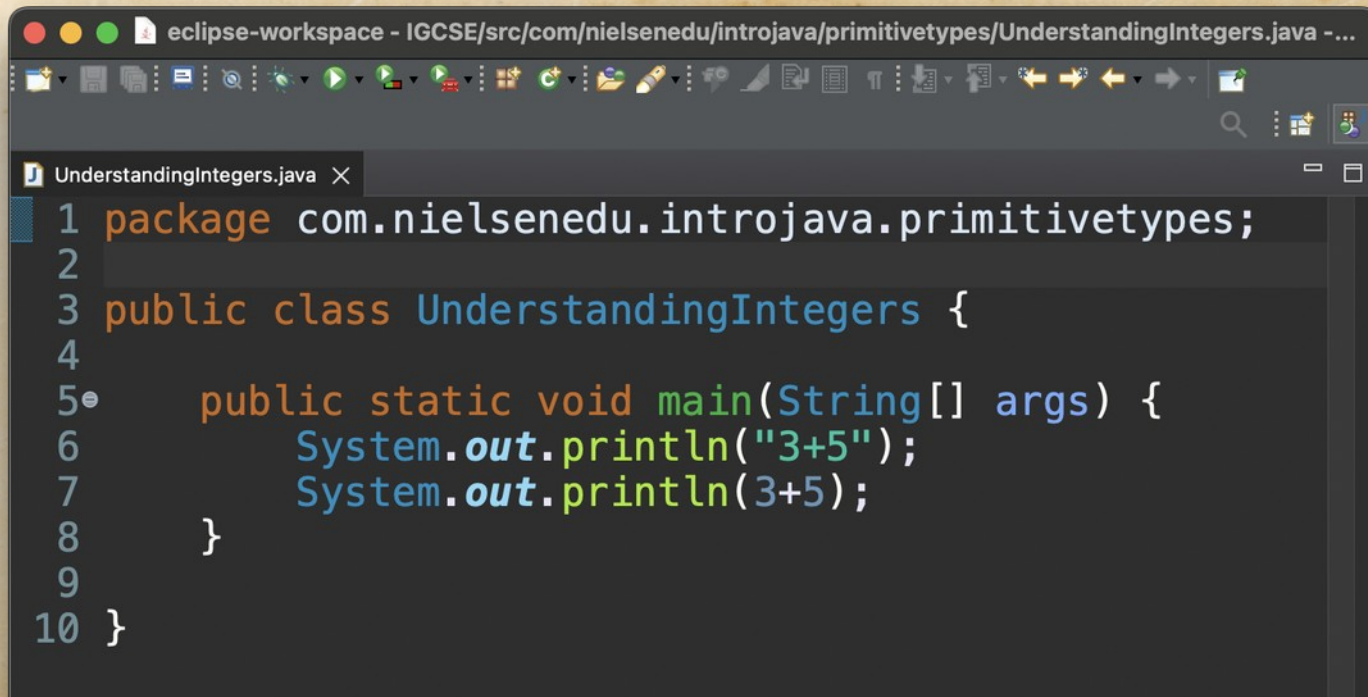
```
1 package com.bjfiles.chris.helloworld;  
2  
3 public class HelloWorld {  
4  
5     public static void main(String[] args) {  
6         System.out.println("Hello World!");  
7     }  
8  
9  
10 }
```

Below the editor, the 'Console' tab is active, showing the output: '<terminated> HelloWorld [Java Application] /Users/christophernielsen/.p2/pool/plugins/org.eclipse.justj.openjdk.h' followed by 'Hello World!'.



# Java **int** Type

- Create a new class named “UnderstandingIntegers”
  - What output do you expect from the program, below?

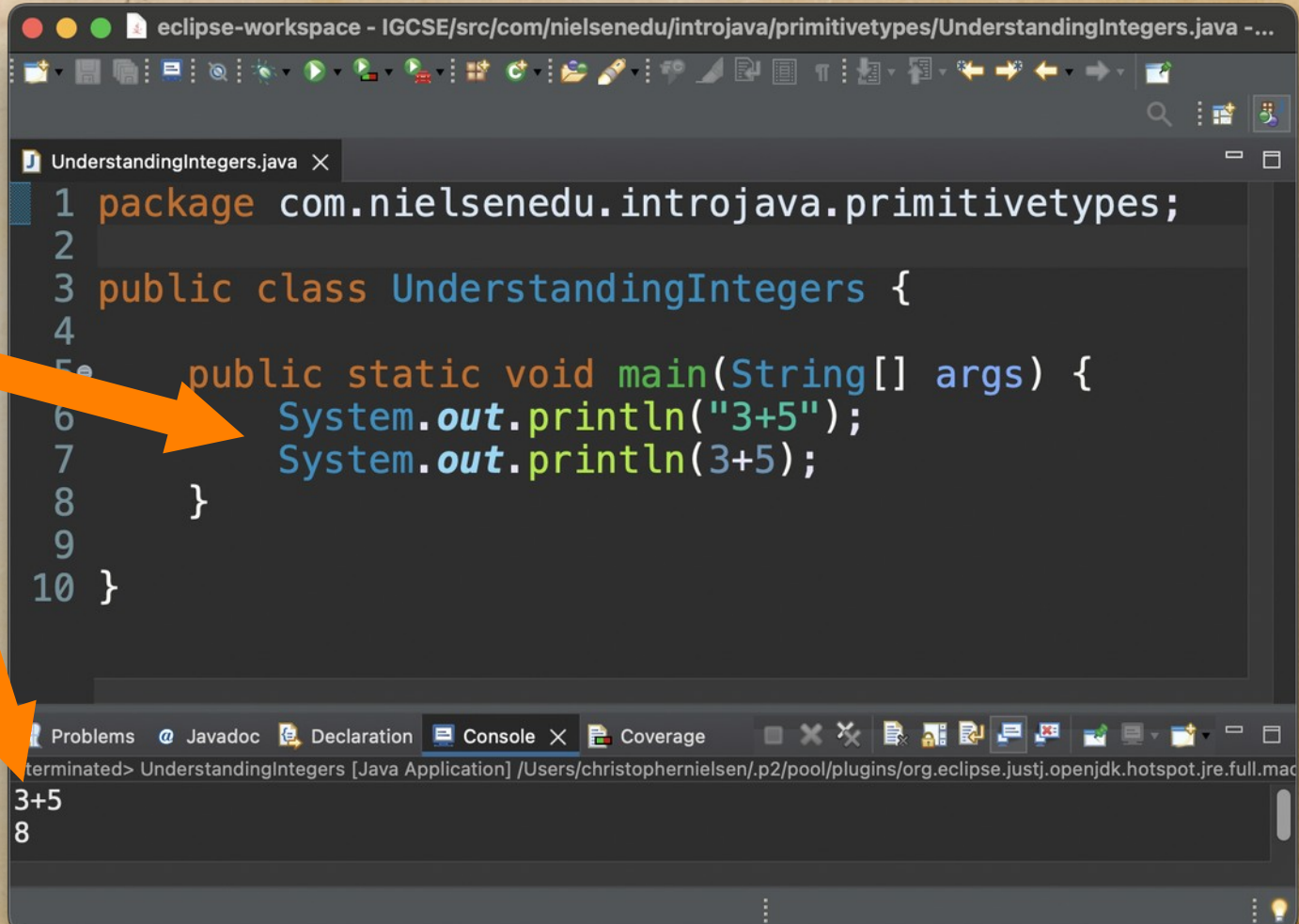


```
eclipse-workspace - IGCSE/src/com/nielsenedu/introjava/primitivetypes/UnderstandingIntegers.java -...  
UnderstandingIntegers.java  
1 package com.nielsenedu.introjava.primitivetypes;  
2  
3 public class UnderstandingIntegers {  
4  
5     public static void main(String[] args) {  
6         System.out.println("3+5");  
7         System.out.println(3+5);  
8     }  
9  
10 }
```



# Java **int** Type

Compare



```
1 package com.nielsenedu.introjava.primitivetypes;
2
3 public class UnderstandingIntegers {
4
5     public static void main(String[] args) {
6         System.out.println("3+5");
7         System.out.println(3+5);
8     }
9
10 }
```

The screenshot shows the Eclipse IDE with a Java file named `UnderstandingIntegers.java`. The code defines a package `com.nielsenedu.introjava.primitivetypes` and a public class `UnderstandingIntegers`. Inside the class, there is a `main` method that prints two lines: `System.out.println("3+5");` and `System.out.println(3+5);`. The IDE's console at the bottom shows the output of the program: `3+5` followed by `8`. An orange arrow points from the word `Compare` to the two `println` statements in the code, indicating that the program is comparing the string representation of the expression `3+5` with its integer result `8`.



# Java **int** Type

- Anything enclosed double quotation marks (") is NOT considered a number by Java, it is considered a **String**.
  - We will learn about the **String** type later.
  - `System.out.println("3 + 5")` will print exactly what is found within the quotation marks: 3 + 5.



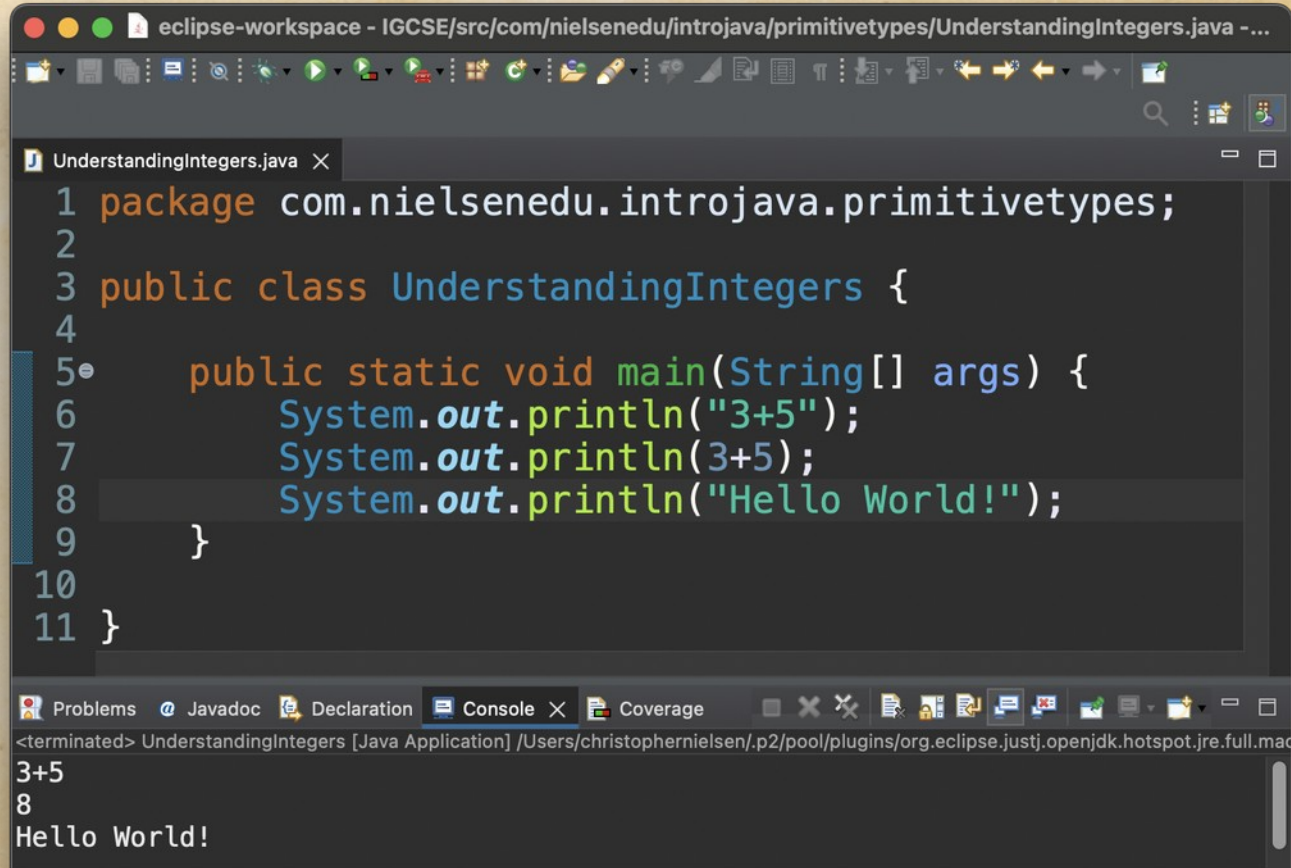
# Java **int** Type

- Integers that are not enclosed double quotation marks (") are considered *integer literals*.
  - Operations, such as addition, can be performed on these integers.
  - `System.out.println(3 + 5)` will perform the addition calculation, then print the result: 8.



# Java **int** Type

- We'll add more lines to our UnderstandingIntegers class...



The screenshot shows the Eclipse IDE interface. The main editor window displays the file `UnderstandingIntegers.java` with the following code:

```
1 package com.nielsenedu.introjava.primitivetypes;
2
3 public class UnderstandingIntegers {
4
5     public static void main(String[] args) {
6         System.out.println("3+5");
7         System.out.println(3+5);
8         System.out.println("Hello World!");
9     }
10
11 }
```

The bottom of the IDE shows the `Console` tab, which displays the output of the program:

```
<terminated> UnderstandingIntegers [Java Application] /Users/christophernielsen/p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64/bin/java
3+5
8
Hello World!
```



# Java **int** Type

- Add each line to your UnderstandingInteger class and observe the output.

```
System.out.println("Hello World!");
```

```
System.out.println("Hello " + "World!");
```

```
System.out.println("Hello " + World);
```

```
System.out.println("3 + 5");
```

```
System.out.println("3" + "5");
```

```
System.out.println(3 + 5);
```



# Java **int** Type

- Add each line to your UnderstandingInteger class and observe the output.

```
System.out.println("Hello World!");
```

```
System.out.println("Hello " + "World!");
```

```
System.out.println("Hello " + World);
```

```
System.out.println("3 + 5");
```

```
System.out.println("3" + "5");
```

```
System.out.println(3 + 5);
```

Hello World!

Hello World!

3 + 5

35

8



# Java **int** Type

```
System.out.println("Hello " + 3);
```

```
System.out.println("Hello " + 3 + 5);
```

```
System.out.println("Hello " + (3 + 5) );
```

```
System.out.println(3 + 5 + " Hello");
```

```
System.out.println(3 + (5 + " Hello") );
```



# Java **int** Type

```
System.out.println("Hello " + 3);
```

```
System.out.println("Hello " + 3 + 5);
```

```
System.out.println("Hello " + (3 + 5) );
```

```
System.out.println(3 + 5 + " Hello");
```

```
System.out.println(3 + (5 + " Hello") );
```

Hello 3

Hello 35

Hello 8

8 Hello

35 Hello

**The addition operation is  
performed from left to right.**



# Primitive Types

Integers