

This lab has **five (5)** checkpoints.

Learning Outcomes

Upon completing this lab, students should be able to

- Implement to an Interface.
- Some bitwise operations.

Introduction

1. Consider the *bits* that make up an int: if $\text{int } x = 5$, then x is stored as the bit sequence $1\ 0\ 1$ or $1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$ (yes, there are a bunch of leading zeros; we are going to ignore those for the moment).

Here are some bitwise operation examples. The last couple use a space between groups of four bits to make the values easier to read.

1010 & 1100	1000	bitwise and
1010 1100	1110	bitwise or
1010 ^ 1100	0110	bitwise xor
~1010	0101	bitwise not
1010 1100 << 3	01100 000	logical shift left
1010 1100 >> 2	0010 1011	logical shift right

- (a) Convert 100_{10} to *binary* in 8 bits.

Solution: 0110 0100

- i. Shift the bits to the right 3 places.

Solution: 0000 1100

- ii. And the value with 1010 1010.

Solution:

	0110 0100
&	1010 1010
	0010 0000

- iii. Or the value with 1100 1100.

Solution:

	0110 0100
	1100 1100
	1110 1100

- iv. Negate the value.

Solution: ~0110 0100 = 1001 1011

- (b) Convert 40_{10} to *binary* in 8 bits (use leading zeros).

Solution: 0010 1000

- i. Shift the bits to the left 2 places.

Solution: 1010 0000

- ii. Convert the answer above back to decimal. What does shifting to the left two places do to the value?

Solution: 160_{10} ; times 2^2

- iii. And the value with 0001 1100.

Solution:

	0010 1000
&	0001 1100
	0000 1000

- iv. Or the value with 1011 0111.

Solution:

	0010 1000
	1011 0111
	1011 1111

- v. Negate the value.

Solution: $\sim 0010\ 1000 = 1101\ 0111$

- vi. Write the negated value in **hexadecimal**.

Solution: D7

✓ Show the lab instructor your work and answers.

2. Initialize a Gradle application project in a new directory. Also initialize git in the root of the project.

The name of the project: storage. Starting package: storage.

Copy the provided StorageClient.java to main.java.storage.

Create a database package. Copy the provided Database.java to database. Look at the interface in Database.java.

The client stores a hundred thousand values into the instantiated Database. As you can see in the code, the client right now instantiates a IntDatabase. Implementing this class is your first checkpoint.

The IntDatabase uses an ArrayList of Integer to hold its values. Notice that this may mean you need to translate the add parameter and/or return values for the get* methods.

Implement all of the interface methods listed in Database for IntDatabase, compile and run the client code. If it works, the output should be

After Construction

After Fill

7! is fine.

Run of bits is fine.

Last value is fine.

Are these the same?

>1111101000<

>1111101000<

✓ Show the lab instructor your working code.

3. Now implement `Database` in the `StringDatabase`, a database that uses an `ArrayList` of `String` that stores bit strings instead of integers.

Modify `StorageClient.java` to use `StringDatabase` and get it to compile and run. The expected output is the same.

✓ Show the lab instructor your working code.

4. [Not a separate checkpoint but necessary] Add and commit your full gradle project (if you `git add` just the root directory, `..`, `git` will add all subdirectories recursively).

Create a repository in your Gitea turn-in organization, associate it with your lab work, and push the repo. This will be done at the end of every lab.