

Computer Science: Create Your Own RPG

Day #5

OPPTAG Explorations 2014

Brian Nakayama¹

¹ Department of Computer Science, Iowa State University, Ames, IA 50010, USA

July 10th, 2014

Extends

- When declaring a class, we can say it *extends* another class.

Extends

- When declaring a class, we can say it *extends* another class.
- If a class A extends another class B , we say that class A is a subclass of B .

Extends

- When declaring a class, we can say it *extends* another class.
- If a class *A* extends another class *B*, we say that class *A* is a subclass of *B*.
- Another way to say this is that *A* is *B*'s child, and *B* is *A*'s parent.

Extends

- When declaring a class, we can say it *extends* another class.
- If a class *A* extends another class *B*, we say that class *A* is a subclass of *B*.
- Another way to say this is that *A* is *B*'s child, and *B* is *A*'s parent.
- We can also say that *A* is derived from *B*, inheriting *B*'s methods.

Extends

- When declaring a class, we can say it *extends* another class.
- If a class *A* extends another class *B*, we say that class *A* is a subclass of *B*.
- Another way to say this is that *A* is *B*'s child, and *B* is *A*'s parent.
- We can also say that *A* is derived from *B*, inheriting *B*'s methods.
- A subclass can use any of the methods of its parent class.

Extends

- When declaring a class, we can say it *extends* another class.
- If a class *A* extends another class *B*, we say that class *A* is a subclass of *B*.
- Another way to say this is that *A* is *B*'s child, and *B* is *A*'s parent.
- We can also say that *A* is derived from *B*, inheriting *B*'s methods.
- A subclass can use any of the methods of its parent class.
- A class can only extend one other class. However, a class can extend a class that extends another class that... etc.
- To access the methods of a parent class, use the keyword `super`.

Overriding

- If a subclass declares a method with the same modifiers, name, and arguments as a parent, we say that the subclass *overrides* the method.

Overriding

- If a subclass declares a method with the same modifiers, name, and arguments as a parent, we say that the subclass *overrides* the method.
- `@Override` declares that a method overrides a parent's method. If the method does not override a parent method this will create an error.

Overriding

- If a subclass declares a method with the same modifiers, name, and arguments as a parent, we say that the subclass *overrides* the method.
- `@Override` declares that a method overrides a parent's method. If the method does not override a parent method this will create an error.
- Take for example the person class with a method `work`. We then might have two subclasses, a teacher and a student, that extend person.

Overriding

- If a subclass declares a method with the same modifiers, name, and arguments as a parent, we say that the subclass *overrides* the method.
- `@Override` declares that a method overrides a parent's method. If the method does not override a parent method this will create an error.
- Take for example the person class with a method `work`. We then might have two subclasses, a teacher and a student, that extend person.

Overriding

- Both the student and the teacher are persons. Thus they can do anything other persons can do.

Overriding

- Both the student and the teacher are persons. Thus they can do anything other persons can do.
- When a teacher works, she or he creates homework.
When a student does work, she or he finishes homework.

Overriding

- Both the student and the teacher are persons. Thus they can do anything other persons can do.
- When a teacher works, she or he creates homework.
When a student does work, she or he finishes homework.
- All classes *descend* from the Object class. This means all objects have inherited the methods from the Object class.
Furthermore, in a way null descends from all classes,
though it does not contain any methods.

Different Persons

```
public class Person{  
    public void work(){  
}
```

```
public class Teacher extends Person{  
    public void work(){  
        //Create Homework Here  
    }  
}
```

```
public class Student extends Person{  
    public void work(){  
        //Finish Homework Here  
    }  
}
```

Why Bother?



Using classes, subclasses, and hierarchies of objects simplifies our code by making it 1. shorter and 2. organizing it by type.

Abstract Classes

- Abstract classes require an object to extend them in order to be used.

Abstract Classes

- Abstract classes require an object to extend them in order to be used.
- Abstract classes can have both normal methods (the ones we are used to), as well as abstract methods. Abstract methods *have to* be overridden by the subclass.
- Since the methods must be seen by subclasses in order to be overridden, they cannot be private.

Abstract Classes

- Abstract classes require an object to extend them in order to be used.
- Abstract classes can have both normal methods (the ones we are used to), as well as abstract methods. Abstract methods *have to* be overridden by the subclass.
- Since the methods must be seen by subclasses in order to be overridden, they cannot be private.
- To declare an abstract method, you use the `abstract` keyword followed by the method declaration followed by a semicolon:

```
abstract public void collision(SimpleObject  
    s);
```

Sneak Peak #4

```
public abstract class SimpleObject {  
    protected final int[] off = 0, 0 ;  
    abstract public void collision(SimpleObject s);  
    abstract public void update();  
    abstract public char id();  
    public String getInfo(){  
        return "";  
    }  
    public SimpleObject getCopy(String s){  
        return null;  
    }  
}
```

...

Interfaces

- Like an abstract class, an *interface* must be *implemented* by a subclass.

Interfaces

- Like an abstract class, an *interface* must be *implemented* by a subclass.
- Unlike an abstract class, interfaces *only* contains empty methods that must be overridden, and they only contain static final variables.

Interfaces

- Like an abstract class, an *interface* must be *implemented* by a subclass.
- Unlike an abstract class, interfaces *only* contains empty methods that must be overridden, and they only contain static final variables.
- The keyword *final* makes a variable constant. Final variables cannot change

Interfaces

- Like an abstract class, an *interface* must be *implemented* by a subclass.
- Unlike an abstract class, interfaces *only* contains empty methods that must be overridden, and they only contain static final variables.
- The keyword *final* makes a variable constant. Final variables cannot change
- However, interfaces are useful since you can *implement* more than one.
- When we access objects through shared methods of a parent class or an interface, we call that (inclusion) *polymorphism*.

The KeyListener Class

```
public interface KeyListener extends EventListener {  
  
    public void keyTyped(KeyEvent e);  
  
    public void keyPressed(KeyEvent e);  
  
    public void keyReleased(KeyEvent e);  
}
```

Interfaces can extend (not implement) other interfaces.
Interfaces cannot extend classes.

Find the Abstract Class Errors!

```
public abstract class animal {  
    abstract int population = 0;  
  
    private abstract void eat(food f);  
    public abstract static void sleep();  
    protected int getPopulation();  
    public abstract void makeNoise(int noise);  
  
    private void sayHello(){  
        System.out.println("I am an animal!");  
    }  
    public abstract void move(){  
        //Move code here...  
    }  
}
```

Find the Interface Errors!

```
public interface updateListener implements updater {  
  
    public final FASTUPDATE = 5;  
    String s;  
  
    public abstract void updateEvent1(int updateID);  
    private void autoUpdate();  
    public void updateEvent2(int updateID);  
}
```

Find the Interface Errors!

```
public interface updateListener implements updater {  
  
    public final FASTUPDATE = 5;  
    String s;  
  
    public abstract void updateEvent1(int updateID);  
    private void autoUpdate();  
    public void updateEvent2(int updateID);  
}
```

If we fixed all the errors in the above code, what methods would a class that implements updateListener have to override?

Introduction

- So far we know the following operators for integers: +, -, *, /, %, and =.
- In addition to these there are bitwise operators: ^, &, |, ~, >>, >>>, and <<.
- Finally there is one *ternary* operator ?:.. It is shorthand for an if else statement.

XOR and AND

- Exclusive OR or XOR(\wedge) outputs a '1' if one bit is a '1' and the other is a '0':

11001101

10010100

01011001

XOR and AND

- Exclusive OR or XOR(\wedge) outputs a '1' if one bit is a '1' and the other is a '0':

11001101

10010100

01011001

- AND(&) outputs a '1' if both bits are a '1':

01100110

10101010

00100010

OR and NOT

- OR(|) outputs a '1' if at least one bit is a '1':

10001101

10100100

10101101

OR and NOT

- OR($|$) outputs a '1' if at least one bit is a '1':

10001101

10100100

10101101

- NOT(\sim) "flips" the bits:

10011010

01100101

Bit Shift

- Shift-left(<<) moves all bits to the left padding with a '0':
(shifting by 1)

10011011

00110110

Bit Shift

- Shift-left(<<) moves all bits to the left padding with a '0': (shifting by 1)

10011011

00110110

- Logical Shift-right(>>) and Arithmetic Shift-right(>>>) moves all bits to the right. Logical shift right pads with a '0' while arithmetic shift copies the leading bit: (shifting by 1)

01001101

00101010

Playing with Bitwise Operators

```
public class Bits {  
  
    public static int a = 5;  
    public static int b = 27;  
    public static int c = 17;  
    public static int d = 14;  
  
    public static void calculate(){  
        int e = b & c;  
        int f = b | c;  
        int g = b >> 1;  
        int h = d & (~c);  
        int i = a ^ b;  
        int j = b << 3;  
        int k = b & ((c | d) >> 2);  
        int l = ((~a) & c) ^ ((~b) & d);  
        int m = (a << (a - 1)) ^ b;  
    }  
}
```

What is a in binary?

Playing with Bitwise Operators

```
public class Bits {  
  
    public static int a = 5;  
    public static int b = 27;  
    public static int c = 17;  
    public static int d = 14;  
  
    public static void calculate(){  
        int e = b & c;  
        int f = b | c;  
        int g = b >> 1;  
        int h = d & (~c);  
        int i = a ^ b;  
        int j = b << 3;  
        int k = b & ((c | d) >> 2);  
        int l = ((~a) & c) ^ ((~b) & d);  
        int m = (a << (a - 1)) ^ b;  
    }  
}
```

What is b in binary?

Playing with Bitwise Operators

```
public class Bits {  
  
    public static int a = 5;  
    public static int b = 27;  
    public static int c = 17;  
    public static int d = 14;  
  
    public static void calculate(){  
        int e = b & c;  
        int f = b | c;  
        int g = b >> 1;  
        int h = d & (~c);  
        int i = a ^ b;  
        int j = b << 3;  
        int k = b & ((c | d) >> 2);  
        int l = ((~a) & c) ^ ((~b) & d);  
        int m = (a << (a - 1)) ^ b;  
    }  
}
```

What is c in binary?

Playing with Bitwise Operators

```
public class Bits {  
  
    public static int a = 5;  
    public static int b = 27;  
    public static int c = 17;  
    public static int d = 14;  
  
    public static void calculate(){  
        int e = b & c;  
        int f = b | c;  
        int g = b >> 1;  
        int h = d & (~c);  
        int i = a ^ b;  
        int j = b << 3;  
        int k = b & ((c | d) >> 2);  
        int l = ((~a) & c) ^ ((~b) & d);  
        int m = (a << (a - 1)) ^ b;  
    }  
}
```

What is d in binary?

Playing with Bitwise Operators

```
public class Bits {  
  
    public static int a = 5;  
    public static int b = 27;  
    public static int c = 17;  
    public static int d = 14;  
  
    public static void calculate(){  
        int e = b & c;  
        int f = b | c;  
        int g = b >> 1;  
        int h = d & (~c);  
        int i = a ^ b;  
        int j = b << 3;  
        int k = b & ((c | d) >> 2);  
        int l = ((~a) & c) ^ ((~b) & d);  
        int m = (a << (a - 1)) ^ b;  
    }  
}
```

What is e in binary?

Playing with Bitwise Operators

```
public class Bits {  
  
    public static int a = 5;  
    public static int b = 27;  
    public static int c = 17;  
    public static int d = 14;  
  
    public static void calculate(){  
        int e = b & c;  
        int f = b | c;  
        int g = b >> 1;  
        int h = d & (~c);  
        int i = a ^ b;  
        int j = b << 3;  
        int k = b & ((c | d) >> 2);  
        int l = ((~a) & c) ^ ((~b) & d);  
        int m = (a << (a - 1)) ^ b;  
    }  
}
```

What is f in binary?

Playing with Bitwise Operators

```
public class Bits {  
  
    public static int a = 5;  
    public static int b = 27;  
    public static int c = 17;  
    public static int d = 14;  
  
    public static void calculate(){  
        int e = b & c;  
        int f = b | c;  
        int g = b >> 1;  
        int h = d & (~c);  
        int i = a ^ b;  
        int j = b << 3;  
        int k = b & ((c | d) >> 2);  
        int l = ((~a) & c) ^ ((~b) & d);  
        int m = (a << (a - 1)) ^ b;  
    }  
}
```

What is g in binary?

Playing with Bitwise Operators

```
public class Bits {  
  
    public static int a = 5;  
    public static int b = 27;  
    public static int c = 17;  
    public static int d = 14;  
  
    public static void calculate(){  
        int e = b & c;  
        int f = b | c;  
        int g = b >> 1;  
        int h = d & (~c);  
        int i = a ^ b;  
        int j = b << 3;  
        int k = b & ((c | d) >> 2);  
        int l = ((~a) & c) ^ ((~b) & d);  
        int m = (a << (a - 1)) ^ b;  
    }  
}
```

What is h in binary?

Playing with Bitwise Operators

```
public class Bits {  
  
    public static int a = 5;  
    public static int b = 27;  
    public static int c = 17;  
    public static int d = 14;  
  
    public static void calculate(){  
        int e = b & c;  
        int f = b | c;  
        int g = b >> 1;  
        int h = d & (~c);  
        int i = a ^ b;  
        int j = b << 3;  
        int k = b & ((c | d) >> 2);  
        int l = ((~a) & c) ^ ((~b) & d);  
        int m = (a << (a - 1)) ^ b;  
    }  
}
```

What is i in binary?

Playing with Bitwise Operators

```
public class Bits {  
  
    public static int a = 5;  
    public static int b = 27;  
    public static int c = 17;  
    public static int d = 14;  
  
    public static void calculate(){  
        int e = b & c;  
        int f = b | c;  
        int g = b >> 1;  
        int h = d & (~c);  
        int i = a ^ b;  
        int j = b << 3;  
        int k = b & ((c | d) >> 2);  
        int l = ((~a) & c) ^ ((~b) & d);  
        int m = (a << (a - 1)) ^ b;  
    }  
}
```

What is j in binary?

Playing with Bitwise Operators

```
public class Bits {  
  
    public static int a = 5;  
    public static int b = 27;  
    public static int c = 17;  
    public static int d = 14;  
  
    public static void calculate(){  
        int e = b & c;  
        int f = b | c;  
        int g = b >> 1;  
        int h = d & (~c);  
        int i = a ^ b;  
        int j = b << 3;  
        int k = b & ((c | d) >> 2);  
        int l = ((~a) & c) ^ ((~b) & d);  
        int m = (a << (a - 1)) ^ b;  
    }  
}
```

What is k in binary?

Playing with Bitwise Operators

```
public class Bits {  
  
    public static int a = 5;  
    public static int b = 27;  
    public static int c = 17;  
    public static int d = 14;  
  
    public static void calculate(){  
        int e = b & c;  
        int f = b | c;  
        int g = b >> 1;  
        int h = d & (~c);  
        int i = a ^ b;  
        int j = b << 3;  
        int k = b & ((c | d) >> 2);  
        int l = ((~a) & c) ^ ((~b) & d);  
        int m = (a << (a - 1)) ^ b;  
    }  
}
```

What is l in binary?

Playing with Bitwise Operators

```
public class Bits {  
  
    public static int a = 5;  
    public static int b = 27;  
    public static int c = 17;  
    public static int d = 14;  
  
    public static void calculate(){  
        int e = b & c;  
        int f = b | c;  
        int g = b >> 1;  
        int h = d & (~c);  
        int i = a ^ b;  
        int j = b << 3;  
        int k = b & ((c | d) >> 2);  
        int l = ((~a) & c) ^ ((~b) & d);  
        int m = (a << (a - 1)) ^ b;  
    }  
}
```

What is m in binary?

Last Chance for Points

Define the following keywords:

public

Last Chance for Points

Define the following keywords:

public
final

Last Chance for Points

Define the following keywords:

public
final
static

Last Chance for Points

Define the following keywords:

public
final
static
private

Last Chance for Points

Define the following keywords:

public
final
static
private
protected

Last Chance for Points

Define the following keywords:

public
final
static
private
protected
void

Last Chance for Points

Define the following keywords:

public
final
static
private
protected
void
abstract

Last Chance for Points

Define the following keywords:

public
final
static
private
protected
void
abstract
class

Last Chance for Points

Define the following keywords:

public
final
static
private
protected
void
abstract
class
interface

Last Chance for Points

Define the following keywords:

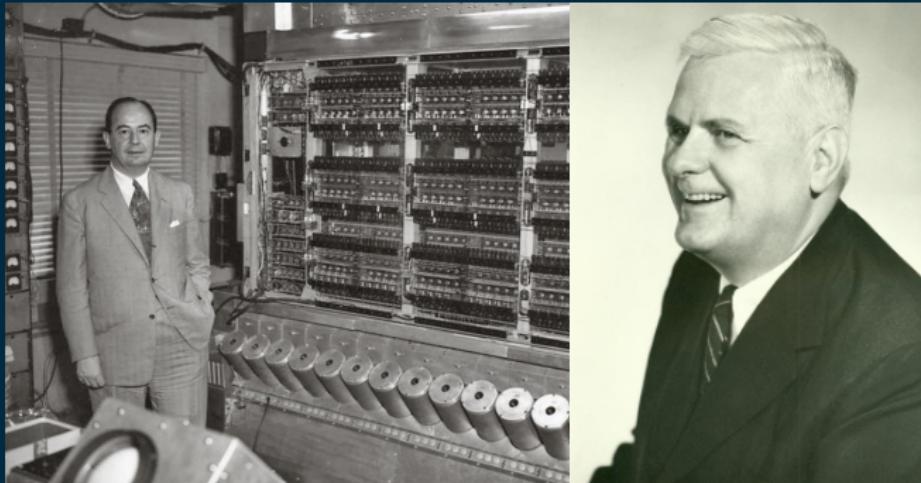
public
final
static
private
protected
void
abstract
class
interface
implements

Last Chance for Points

Define the following keywords:

public
final
static
private
protected
void
abstract
class
interface
implements
extends

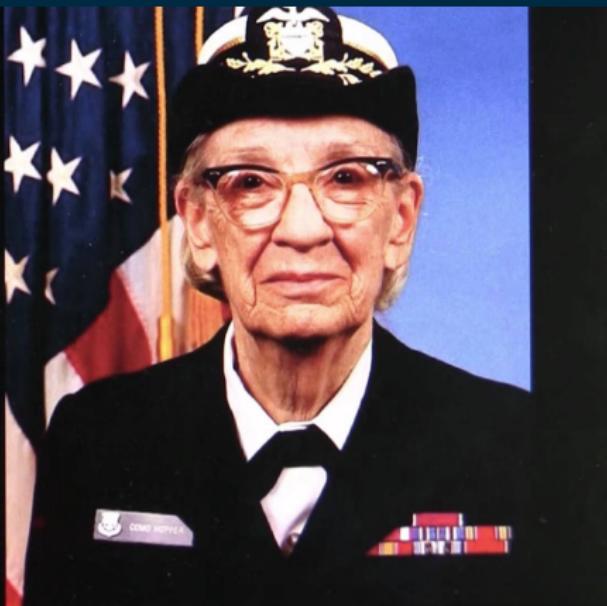
Jon von Neumann and Alonzo Church



Grace Hopper

*"Nobody believed
that I had a
running compiler
and nobody would
touch it. They told
me computers
could only do
arithmetic."*

Grace Hopper



Abstract and Interface
oooooooo
oooo

Bitwise Operations
ooooooo

The Interesting Lives of Computer Scientists
oo•oooooooo

Some Games I Enjoy
oooo

Grace Hopper

92

9/9

0800 Antran started ✓ { 1.2700 9.032 847 025
1000 stopped - antran ✓ 9.037 846 795 connect
13'00 (033) MP - MC ~~1.2700~~ 4.615 925059 (-)
033 PRO 2 2.13047645
connect 2.13067645
Relays 6-2 in 033 failed special speed test
in relay 10.00 test.

Relay 2145
Relay 3370

1100 Started Cosine Tape (Sine check)

1525 Started Multi. Adder Test.

1545  Relay #70 Panel F
(moth) in relay.

1620 First actual case of bug being found.
1700 closed form.

Grace Hopper



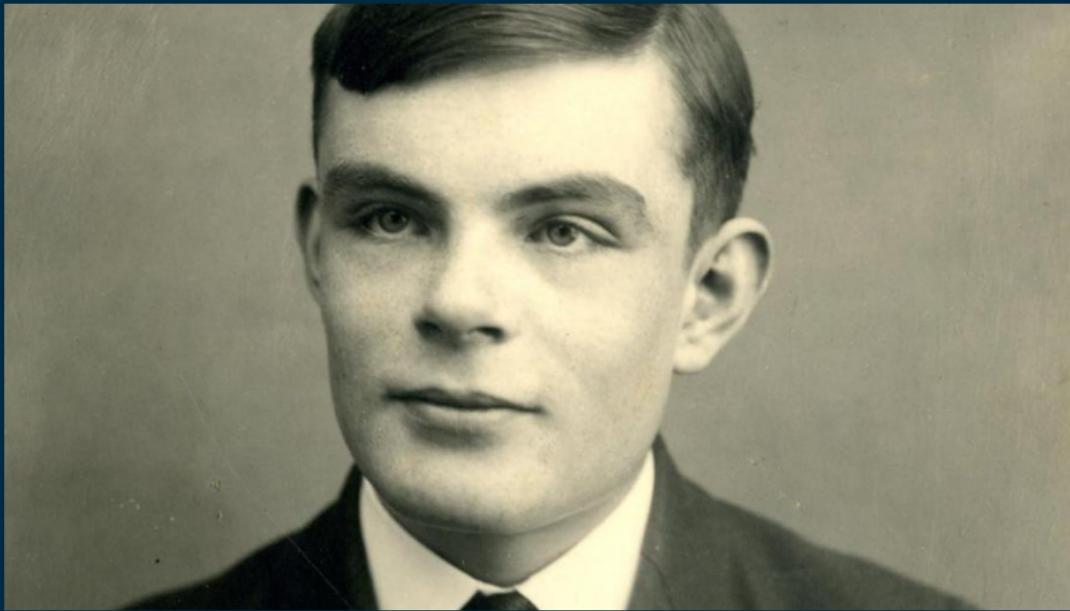
Abstract and Interface
oooooooo
oooo

Bitwise Operations
ooooooo

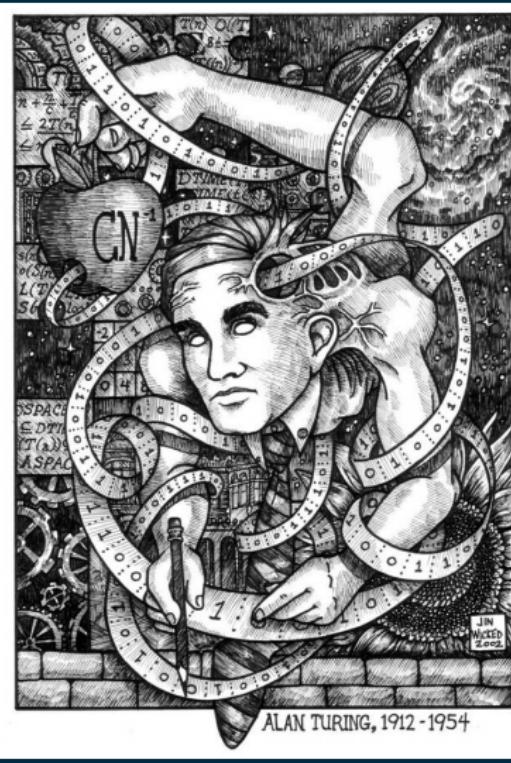
The Interesting Lives of Computer Scientists
oooo●oooo

Some Games I Enjoy
oooo

Alan Turing



Alan Turing



Abstract and Interface
oooooooo
oooo

Bitwise Operations
ooooooo

The Interesting Lives of Computer Scientists
oooooooo●ooo

Some Games I Enjoy
oooo

Ada Lovelace



Ada Lovelace



Ada Lovelace



Ada Lovelace



Abstract and Interface
oooooooo
oooo

Bitwise Operations
ooooooo

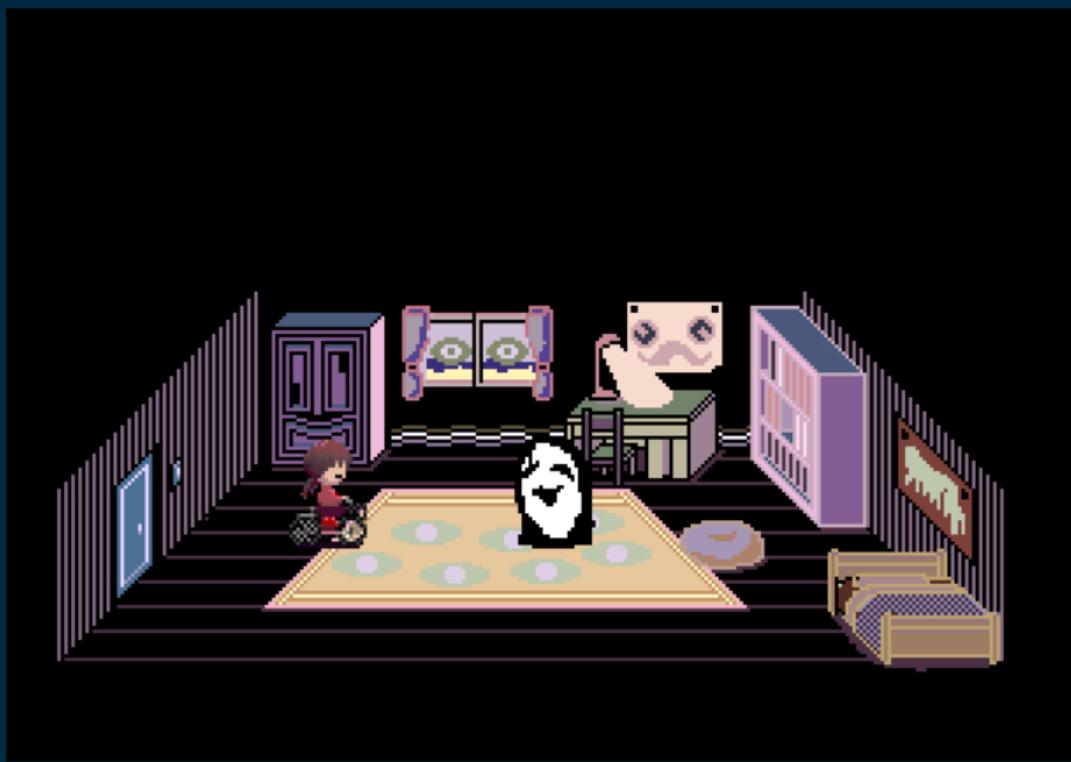
The Interesting Lives of Computer Scientists
oooooooooooo

Some Games I Enjoy
●oooo

|b



Yume Nikki



Abstract and Interface
oooooooo
oooo

Bitwise Operations
ooooooo

The Interesting Lives of Computer Scientists
oooooooooooo

Some Games I Enjoy
oooo

Voxatron



Voxatron

