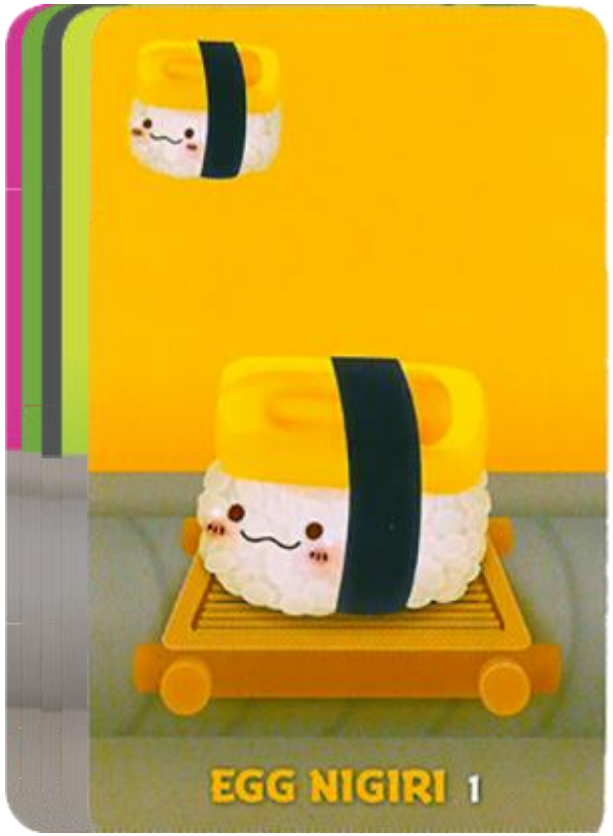
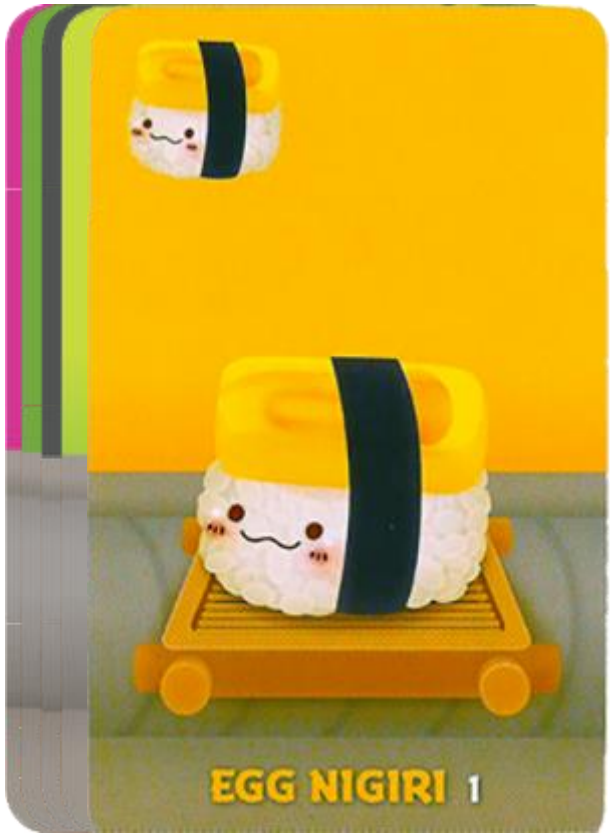
A Stack of Objects

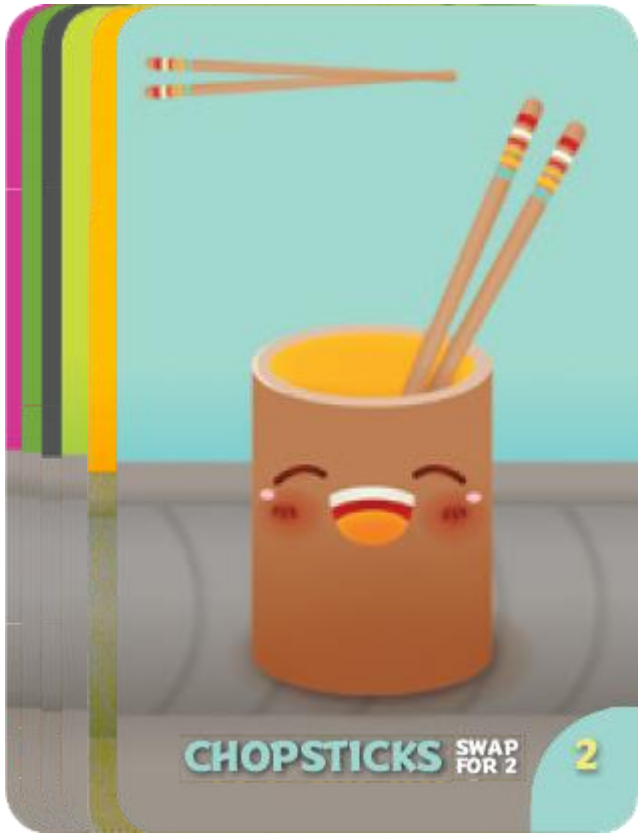


A card game often has a deck.

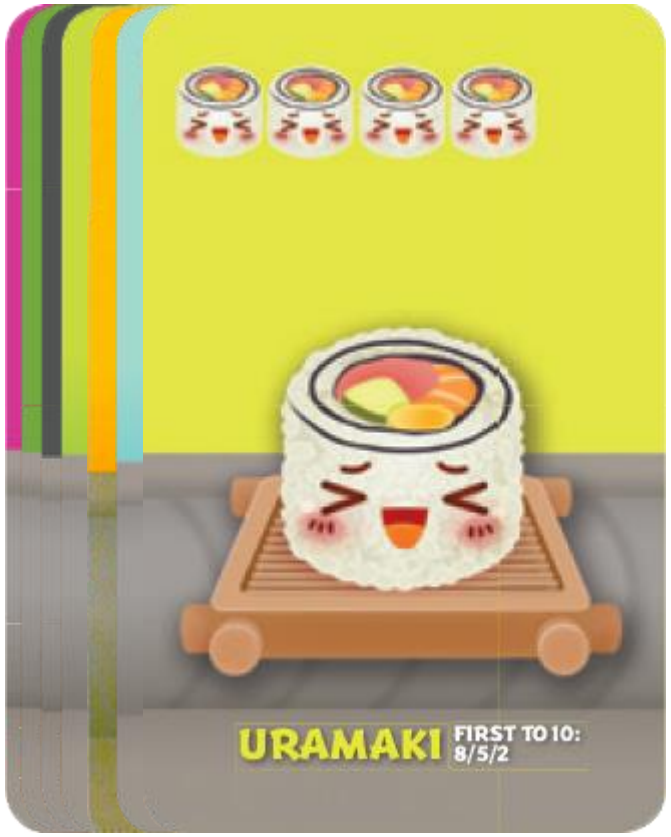
And a deck behaves like a stack.



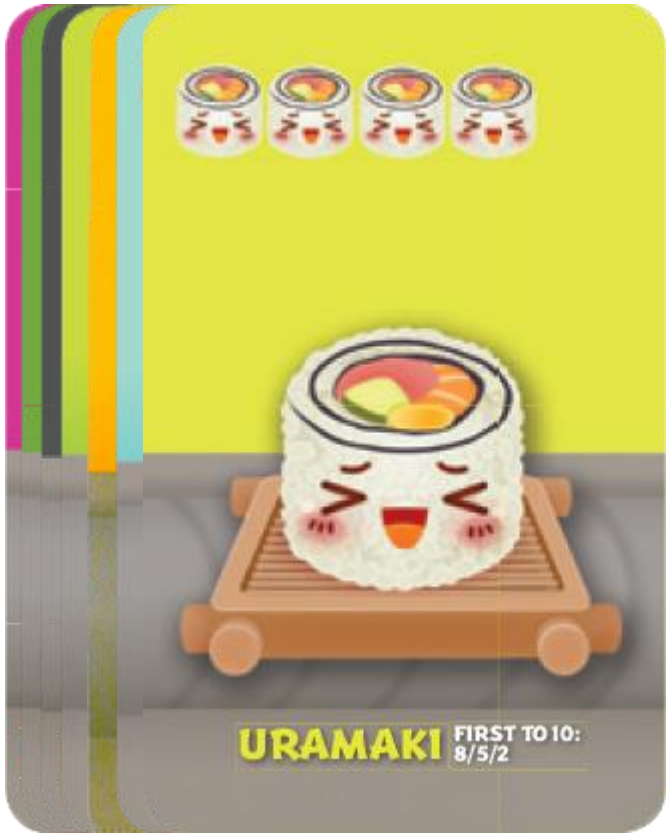
Push = add



Push = add

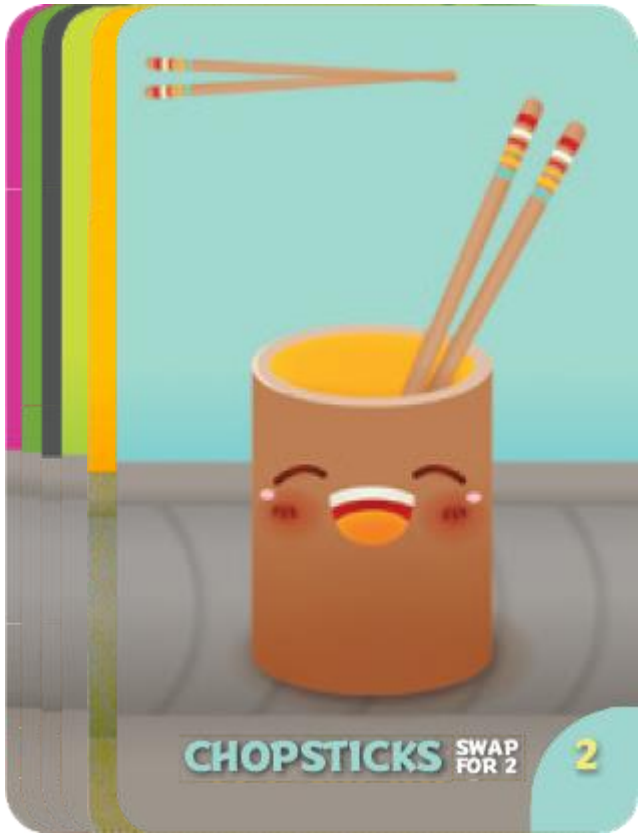


Deal 4 cards = pop 4 cards

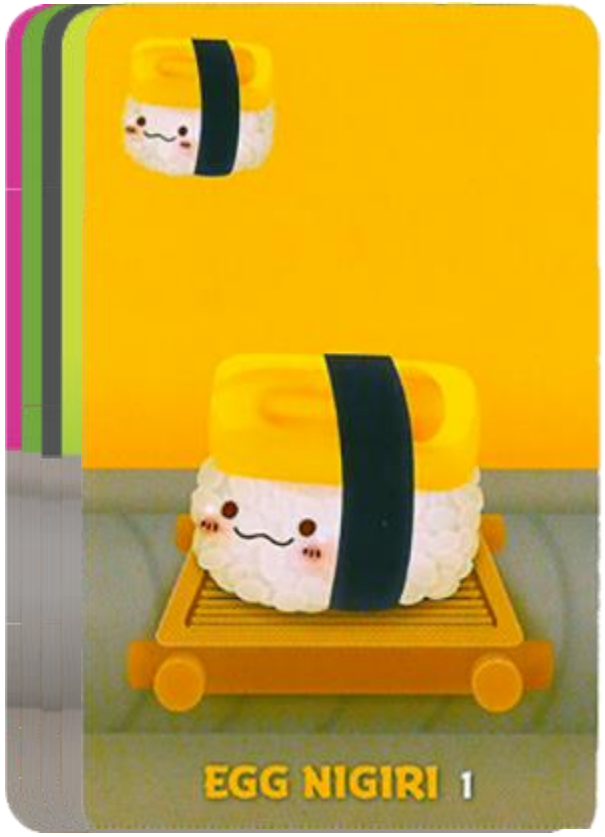


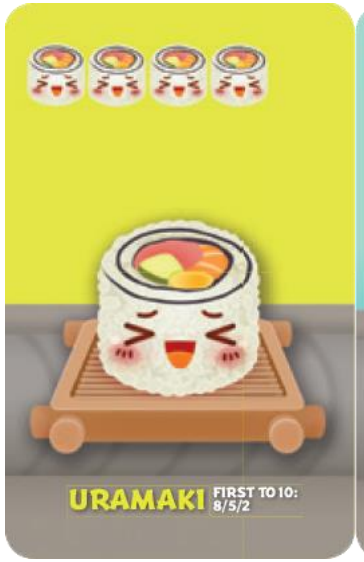
Pop 1.

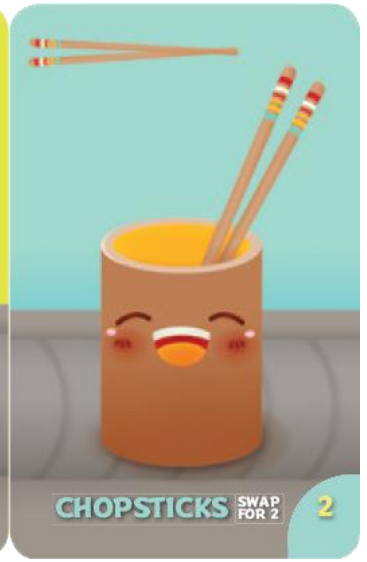
Pop 2.



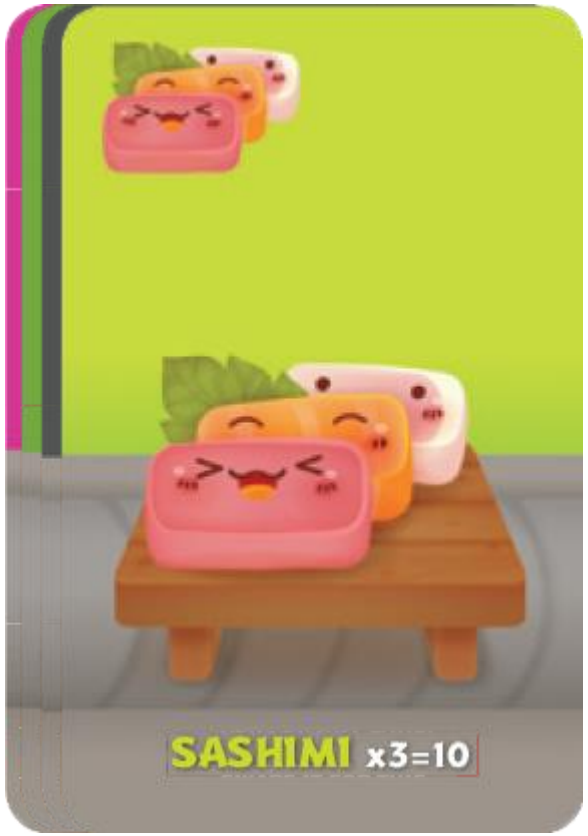
Pop 3.

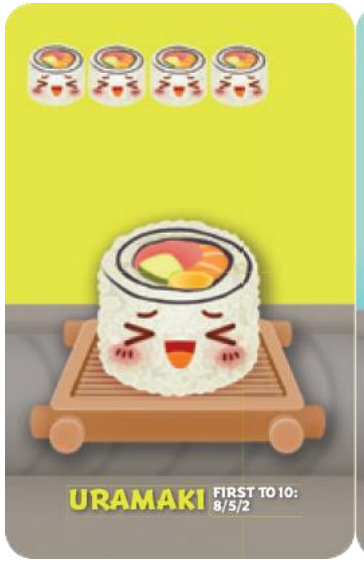






Pop 4.

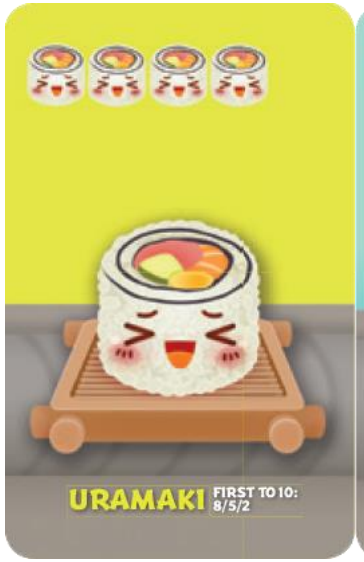

SASHIMI x3=10


URAMAKI FIRST TO 10: 8/5/2


CHOPSTICKS SWAP FOR 2
2


EGG NIGIRI 1

Hand is dealt.



I need a card object.
Each card will be added to my card deck

Num: 1, 2, or 3
Name: Maki Roll
Image: Maki.png = image #6
Msg: Most: 6/3

```
public class Sushi {

  private int num;
  private String name;

  public Sushi () {
    num = 2;
    name = "Maki Roll";
  }

  public Sushi (int n, String na) {
    num = n;
    name = na;
  }

  public String getName () {
    return name;
  }

  public int getNum () {
    return num;
  }
```

Instance variables

```
  public int getPic () {
    if (name.equals ("Maki Roll"))
      return 6;
    else if (name.equals ("Spoon"))
      return 5;
    else if (name.equals ("Chopsticks"))
      return 4;
    else if (name.equals ("Uramaki"))
      return 3;
    else if (name.equals ("Egg Nigiri"))
      return 2;
    else //if (name.equals ("Sashimi"))
      return 1;
  }

  public String getMsg ()  {
    if (name.equals ("Maki Roll"))
      return "Most: 6/3";
    else if (name.equals ("Spoon"))
      return "Ask for any card, swap left";
    else if (name.equals ("Chopsticks"))
      return "Swap for 2";
    else if (name.equals ("Uramaki"))
      return "First to 10: 8/5/2";
    else if (name.equals ("Egg Nigiri"))
      return "1";
    else //if (name.equals ("Sashimi"))
      return "x3=10";
  }
```

```
  public boolean equals (Sushi s) {
    if (s.getName () == name &&
                    s.getNum () == num)
      return true;
    else
      return false;
  }

  public int compareTo (Sushi s) {
    if (name.compareTo (s.getName()) > 1)
      return 1;
    else if (name.equals (s.getName ()) &&
                          num > s.getNum ())
      return 1;
    else if (s.getName () == name &&
                          s.getNum () == num)
      return 0;
    else
      return -1;
    }

}
```

```
public class Sushi {

  private int num;
  private String name;

  public Sushi () {
    num = 2;
    name = "Maki Roll";
  }

  public Sushi (int n, String na) {
    num = n;
    name = na;
  }

  public String getName () {
    return name;
  }

  public int getNum () {
    return num;
  }
```

Instance variables

Default constructor

Constructor where they pick

```
public int getPic () {
   if (name.equals ("Maki Roll"))
     return 6;
   else if (name.equals ("Spoon"))
     return 5;
   else if (name.equals ("Chopsticks"))
     return 4;
   else if (name.equals ("Uramaki"))
     return 3;
   else if (name.equals ("Egg Nigiri"))
     return 2;
   else //if (name.equals ("Sashimi"))
     return 1;
 }

 public String getMsg ()  {
   if (name.equals ("Maki Roll"))
     return "Most: 6/3";
   else if (name.equals ("Spoon"))
     return "Ask for any card, swap left";
   else if (name.equals ("Chopsticks"))
     return "Swap for 2";
   else if (name.equals ("Uramaki"))
     return "First to 10: 8/5/2";
   else if (name.equals ("Egg Nigiri"))
     return "1";
   else //if (name.equals ("Sashimi"))
     return "x3=10";
 }
```

```
  public boolean equals (Sushi s) {
    if (s.getName () == name &&
                        s.getNum () == num)
      return true;
    else
      return false;
  }

  public int compareTo (Sushi s) {
    if (name.compareTo (s.getName()) > 1)
      return 1;
    else if (name.equals (s.getName ()) &&
                              num > s.getNum ())
      return 1;
    else if (s.getName () == name &&
                              s.getNum () == num)
      return 0;
    else
      return -1;
  }

}
```

```
public class Sushi {

  private int num;
  private String name;

  public Sushi () {
    num = 2;
    name = "Maki Roll";
  }

  public Sushi (int n, String na) {
    num = n;
    name = na;
  }

  public String getName () {
    return name;
  }

  public int getNum () {
    return num;
  }
```

Instance variables

Default constructor

Constructor where they pick

Accessor

Accessor

```
public int getPic () {
   if (name.equals ("Maki Roll"))
     return 6;
   else if (name.equals ("Spoon"))
     return 5;
   else if (name.equals ("Chopsticks"))
     return 4;
   else if (name.equals ("Uramaki"))
     return 3;
   else if (name.equals ("Egg Nigiri"))
     return 2;
   else //if (name.equals ("Sashimi"))
     return 1;
 }

 public String getMsg ()  {
   if (name.equals ("Maki Roll"))
     return "Most: 6/3";
   else if (name.equals ("Spoon"))
     return "Ask for any card, swap left";
   else if (name.equals ("Chopsticks"))
     return "Swap for 2";
   else if (name.equals ("Uramaki"))
     return "First to 10: 8/5/2";
   else if (name.equals ("Egg Nigiri"))
     return "1";
   else //if (name.equals ("Sashimi"))
     return "x3=10";
 }
```

```
  public boolean equals (Sushi s) {
    if (s.getName () == name &&
                        s.getNum () == num)
      return true;
    else
      return false;
  }

  public int compareTo (Sushi s) {
    if (name.compareTo (s.getName()) > 1)
      return 1;
    else if (name.equals (s.getName ()) &&
                              num > s.getNum ())
      return 1;
    else if (s.getName () == name &&
                              s.getNum () == num)
      return 0;
    else
      return -1;
  }

}
```

```
public class Sushi {

  private int num;
  private String name;

  public Sushi () {
    num = 2;
    name = "Maki Roll";
  }

  public Sushi (int n, String na) {
    num = n;
    name = na;
  }

  public String getName () {
    return name;
  }

  public int getNum () {
    return num;
  }
```

Instance variables

Default constructor

Constructor where they pick

Accessor

Accessor

```
public int getPic () {
  if (name.equals ("Maki Roll"))
    return 6;
  else if (name.equals ("Spoon"))
    return 5;
  else if (name.equals ("Chopsticks"))
    return 4;
  else if (name.equals ("Uramaki"))
    return 3;
  else if (name.equals ("Egg Nigiri"))
    return 2;
  else //if (name.equals ("Sashimi"))
    return 1;
}

public String getMsg ()  {
  if (name.equals ("Maki Roll"))
    return "Most: 6/3";
  else if (name.equals ("Spoon"))
    return "Ask for any card, swap left";
  else if (name.equals ("Chopsticks"))
    return "Swap for 2";
  else if (name.equals ("Uramaki"))
    return "First to 10: 8/5/2";
  else if (name.equals ("Egg Nigiri"))
    return "1";
  else //if (name.equals ("Sashimi"))
    return "x3=10";
}
```

Accessor

Accessor

```
public boolean equals (Sushi s) {
  if (s.getName () == name &&
                    s.getNum () == num)
    return true;
  else
    return false;
}

public int compareTo (Sushi s) {
  if (name.compareTo (s.getName()) > 1)
    return 1;
  else if (name.equals (s.getName ()) &&
                          num > s.getNum ())
    return 1;
  else if (s.getName () == name &&
                          s.getNum () == num)
    return 0;
  else
    return -1;
  }

}
```

```
public class Sushi {

  private int num;
  private String name;

  public Sushi () {
    num = 2;
    name = "Maki Roll";
  }

  public Sushi (int n, String na) {
    num = n;
    name = na;
  }

  public String getName () {
    return name;
  }

  public int getNum () {
    return num;
  }
```

Instance variables

Default constructor

Constructor where they pick

Accessor

Accessor

```
public int getPic () {
  if (name.equals ("Maki Roll"))
    return 6;
  else if (name.equals ("Spoon"))
    return 5;
  else if (name.equals ("Chopsticks"))
    return 4;
  else if (name.equals ("Uramaki"))
    return 3;
  else if (name.equals ("Egg Nigiri"))
    return 2;
  else //if (name.equals ("Sashimi"))
    return 1;
}

public String getMsg ()  {
  if (name.equals ("Maki Roll"))
    return "Most: 6/3";
  else if (name.equals ("Spoon"))
    return "Ask for any card, swap left";
  else if (name.equals ("Chopsticks"))
    return "Swap for 2";
  else if (name.equals ("Uramaki"))
    return "First to 10: 8/5/2";
  else if (name.equals ("Egg Nigiri"))
    return "1";
  else //if (name.equals ("Sashimi"))
    return "x3=10";
}
```

Accessor

Accessor

```
public boolean equals (Sushi s) {
  if (s.getName () == name &&
                  s.getNum () == num)
    return true;
  else
    return false;
}

public int compareTo (Sushi s) {
  if (name.compareTo (s.getName()) > 1)
    return 1;
  else if (name.equals (s.getName ()) &&
                       num > s.getNum ())
    return 1;
  else if (s.getName () == name &&
                       s.getNum () == num)
    return 0;
  else
    return -1;
  }
}
```

Facilitator

Facilitator

Sort by name, then by num

Making an ADT of Objects

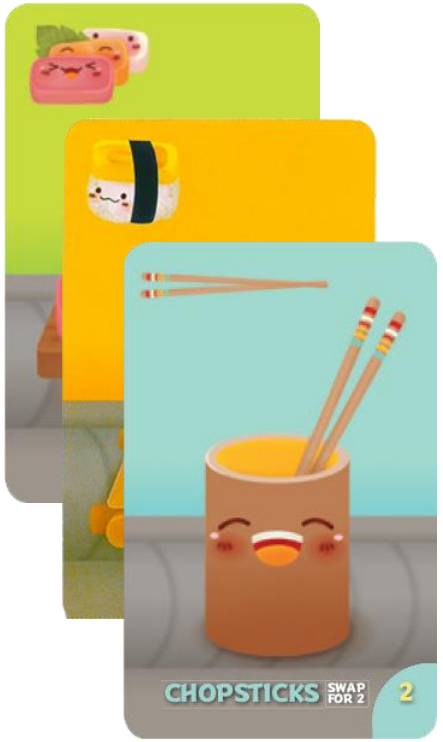
- Objects can build on each other.
- For example, we can make a stack of cards
  or a queue of printer jobs.

Steps

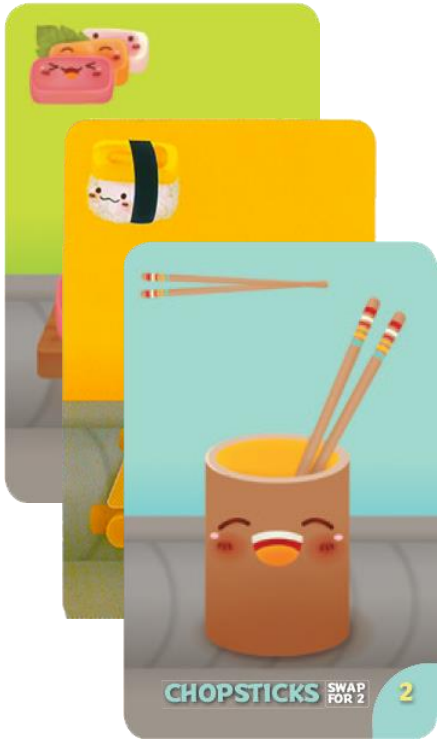
1. Change name of class (eg. Stack -> SushiStack).
2. Change name of constructor
3. Change all “Object” to your Object.
   (eg. Object -> Sushi)

```
public class Stack {
  private int count;
  private Object data[] = new Object [50];

  public Stack () {
    count = 0;
  }

  public void push (Object addMe) {
    data [count] = addMe;
    count++;
  }

  public int size () {
    return count;
  }

  public boolean isFull () {
    return (count == 50);
  }
```

```
  public Object pop () {
    count--;
    return data [count];
  }

  public Object peek () {
    return data [count--];
  }

  public boolean isEmpty () {
    return count == 0;
  }

  public void clear ()  {
    count = 0;
  }
}
```



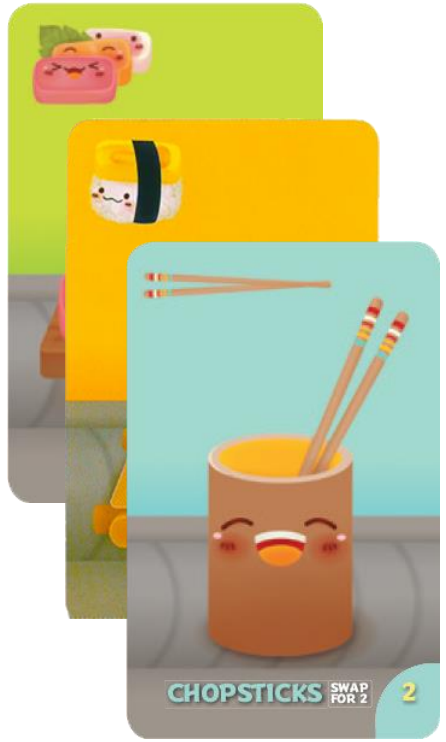
```
public class Stack {
  private int count;
  private Object data[] = new Object [50];

  public Stack () {
    count = 0;
  }

  public void push (Object addMe) {
    data [count] = addMe;
    count++;
  }

  public int size () {
    return count;
  }

  public boolean isFull () {
    return (count == 50);
  }

  public Object pop () {
    count--;
    return data [count];
  }

  public Object peek () {
    return data [count--];
  }

  public boolean isEmpty () {
    return count == 0;
  }

  public void clear ()  {
    count = 0;
  }
}
```



Change class name to distinguish from other Stacks

```
public class SushiStack {
  private int count;
  private Sushi data[] = new Sushi [50];

  public SushiStack () {
    count = 0;
  }

  public void push ( Sushi addMe) {
    data [count] = addMe;
    count++;
  }

  public int size () {
    return count;
  }

  public boolean isFull () {
    return (count == 50);
  }
```

```
  public Sushi pop () {
    count--;
    return data [count];
  }

  public Sushi peek () {
    return data [count--];
  }

  public boolean isEmpty () {
    return count == 0;
  }

  public void clear ()  {
    count = 0;
  }
}
```



Using your ADT of Objects

- Requires 3 classes: (1) Object, (2) ADT of Objects and (3) Runner file
- In the runner; construct an ADT of Objects:

```
SushiStack s = new SushiStack ();
```

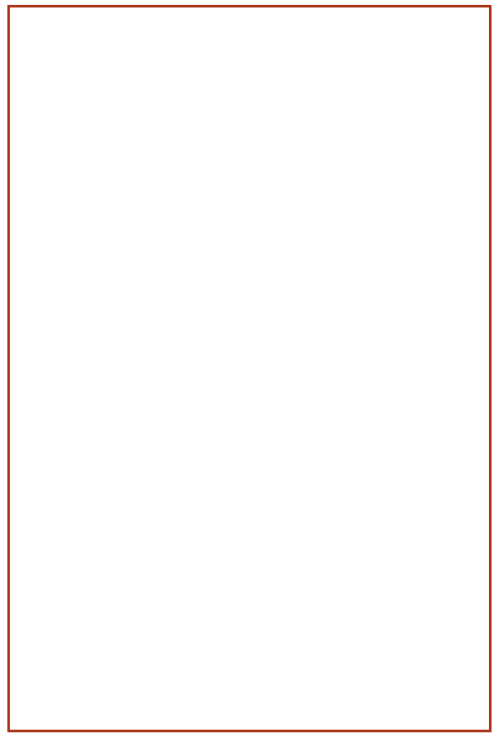
- Make object first and add object:

```
Sushi n = new Sushi (1,"Chopsticks");
 s.push (n);
```

- Or make and add all in one line:

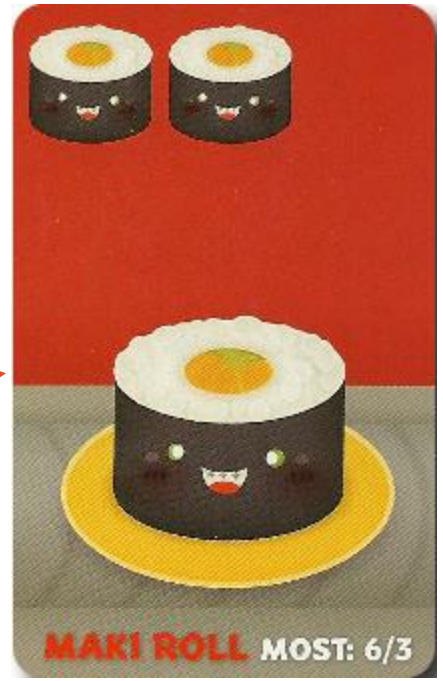
```
s.push (new Sushi (1,"Egg Nigiri"));
```

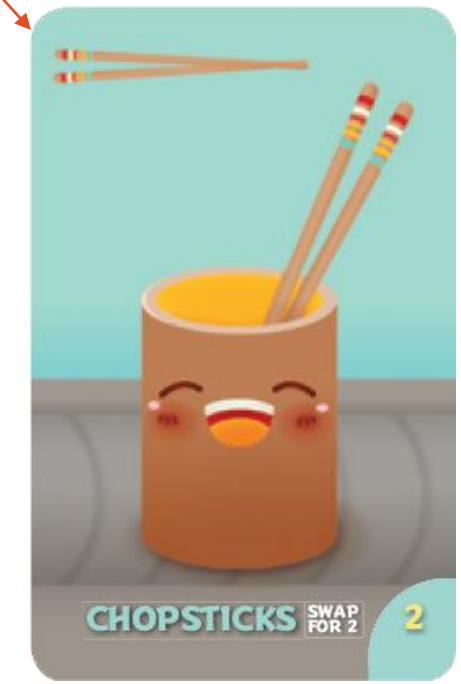
```
public class SushiStackRunner {
   public static void main (String args[]) {
      SushiStack s = new SushiStack ();
      Sushi n = new Sushi ();
      Sushi n2 = new Sushi (1,"Chopsticks");
      s.push (n);
      s.push (n2);
      s.push (new Sushi (1,"Egg Nigiri"));

      Sushi temp = s.pop();
      System.out.println(temp.getName());
   }
}
```



S

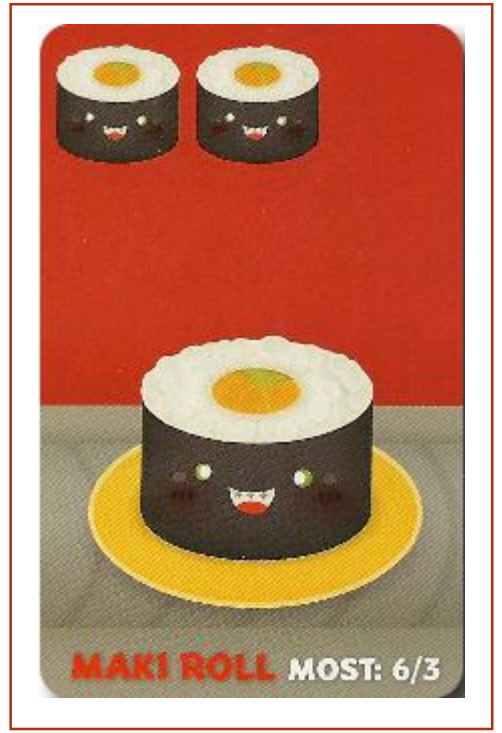
```
public class SushiStackRunner {
   public static void main (String args[]) {
      SushiStack s = new SushiStack ();
      Sushi n = new Sushi ();
      Sushi n2 = new Sushi (1,"Chopsticks");
      s.push (n);
      s.push (n2);
      s.push (new Sushi (1,"Egg Nigiri"));

      Sushi temp = s.pop();
      System.out.println(temp.getName());
   }
}
```





S

```
public class SushiStackRunner {
   public static void main (String args[]) {
      SushiStack s = new SushiStack ();
      Sushi n = new Sushi ();
      Sushi n2 = new Sushi (1,"Chopsticks");
      s.push (n);
      s.push (n2);
      s.push (new Sushi (1,"Egg Nigiri"));

      Sushi temp = s.pop();
      System.out.println(temp.getName());
   }
}
```



S

```
public class SushiStackRunner {
   public static void main (String args[]) {
      SushiStack s = new SushiStack ();
      Sushi n = new Sushi ();
      Sushi n2 = new Sushi (1,"Chopsticks");
      s.push (n);
      s.push (n2);
      s.push (new Sushi (1,"Egg Nigiri"));

      Sushi temp = s.pop();
      System.out.println(temp.getName());
   }
}
```

S

```
public class SushiStackRunner {
   public static void main (String args[]) {
      SushiStack s = new SushiStack ();
      Sushi n = new Sushi ();
      Sushi n2 = new Sushi (1,"Chopsticks");
      s.push (n);
      s.push (n2);
      s.push (new Sushi (1,"Egg Nigiri"));

      Sushi temp = s.pop();
      System.out.println(temp.getName());
   }
}
```

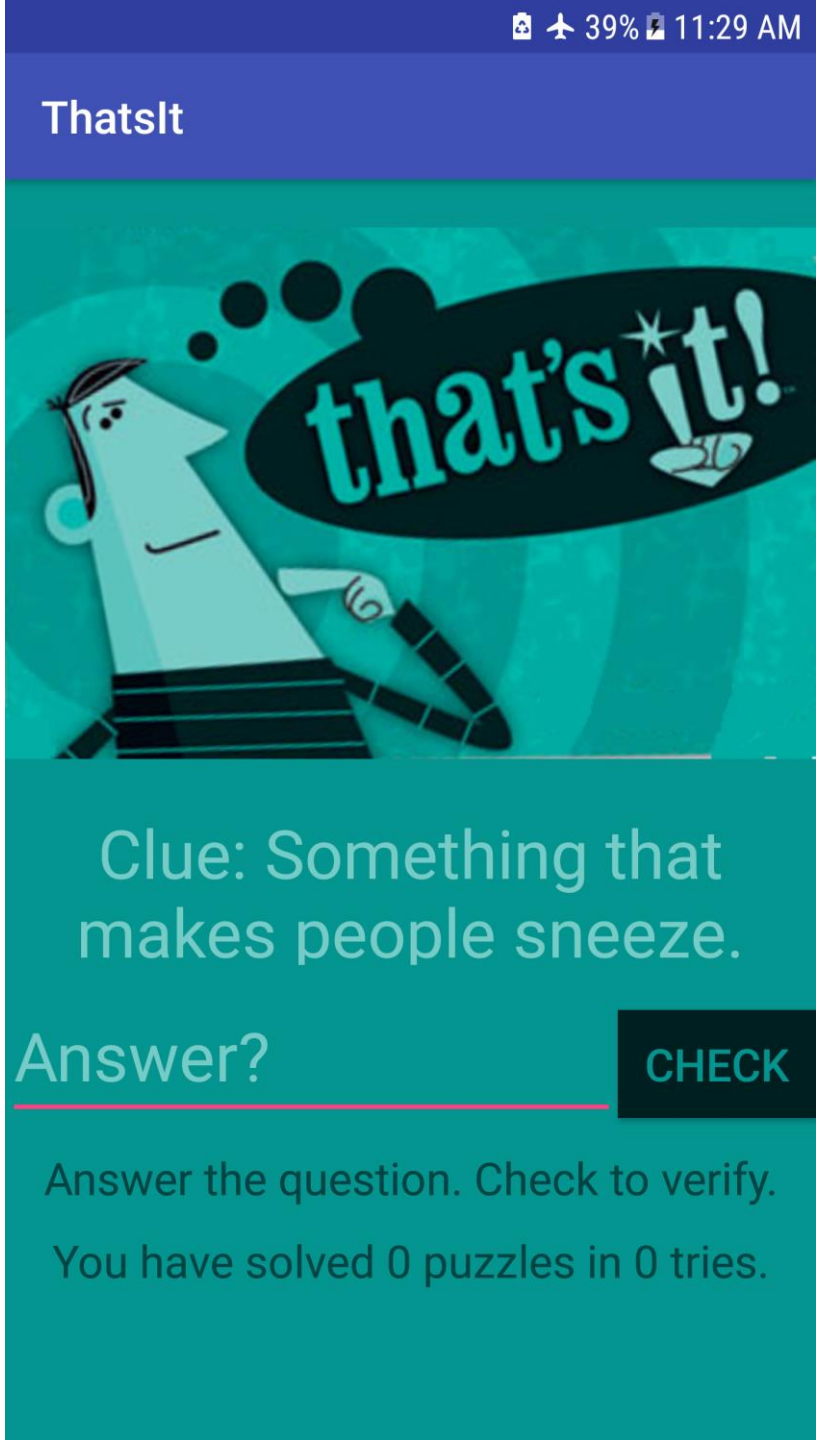
S

```
public class SushiStackRunner {
   public static void main (String args[]) {
      SushiStack s = new SushiStack ();
      Sushi n = new Sushi ();
      Sushi n2 = new Sushi (1,"Chopsticks");
      s.push (n);
      s.push (n2);
      s.push (new Sushi (1,"Egg Nigiri"));

      Sushi temp = s.pop();
      System.out.println(temp.getName());
   }
}
```

S

Final Project

Deck of Cards


39% 11:29 AM
ThatsIt
that's it!
Clue: Something that makes people sneeze.
Answer?
CHECK
Answer the question. Check to verify.
You have solved 0 puzzles in 0 tries.


that's it!
JUST THE RIGHT PARTY GAME
GAMEWRIGHT
A type of entertainer on a cruise ship. MAGICIAN
Something that has a string attached. YO-YO
A type of math. ALGEBRA
A noble gas. ARGON
NIGHTMARE
Something found in a tool box. HAMMER
A take-out food. PIZZA
CRAYON
An animal with whiskers. WALRUS
A place where you might see stars. HOLLYWOOD
POLLEN
FINGER
1
2
4
5
that's it!
GAMEWRIGHT


SPEEDY
WORDS
A QUICK GAME FOR QUICK MINDS!
FoxMind
M
P
R

39% 11:11 AM

Bhatty_SpeedyWords

SPEEDY WORDS

SHUFFLE

F A J

Category:

PLAYER 1: 0

PLAYER 2: 0

PLAYER 3: 0

PLAYER 4: 0

RESET

39% 11:11 AM

Bhatty_SpeedyWords

SPEEDY WORDS

SHUFFLE

E I V

Category: Song

PLAYER 1: 2

PLAYER 2: 1

PLAYER 3: 1

PLAYER 4: 2

Player 4 got a point!

RESET

GAMEWRIGHT®

JOE NAME IT™

NOT YOUR AVERAGE PARTY GAME

Can you name a movie with 2 sequels? How about a music group with 3 members? Test your numbers know-all in this fast-playing "name it to claim it" party game. Draw a category card and then fill in the blank with the number you rolled. Be the first to name something that fits and you're smarter than the average Joe!

2+ PLAYERS | AGES 12 & UP
200 CARDS | NUMBER DIE | RULES OF PLAY
PLAYING TIME: 15 MINUTES

37% 11:16 AM

Kaur_JoeNameIt

JOE NAME IT™

RESET SAVE OPEN

JOE: 0

ANY JOE

Name a word with __ vowel(s).

AMANDA : 1

NANCY : 0

37% 11:16 AM

Kaur_JoeNameIt

JOE NAME IT™

RESET SAVE OPEN

JOE: 0

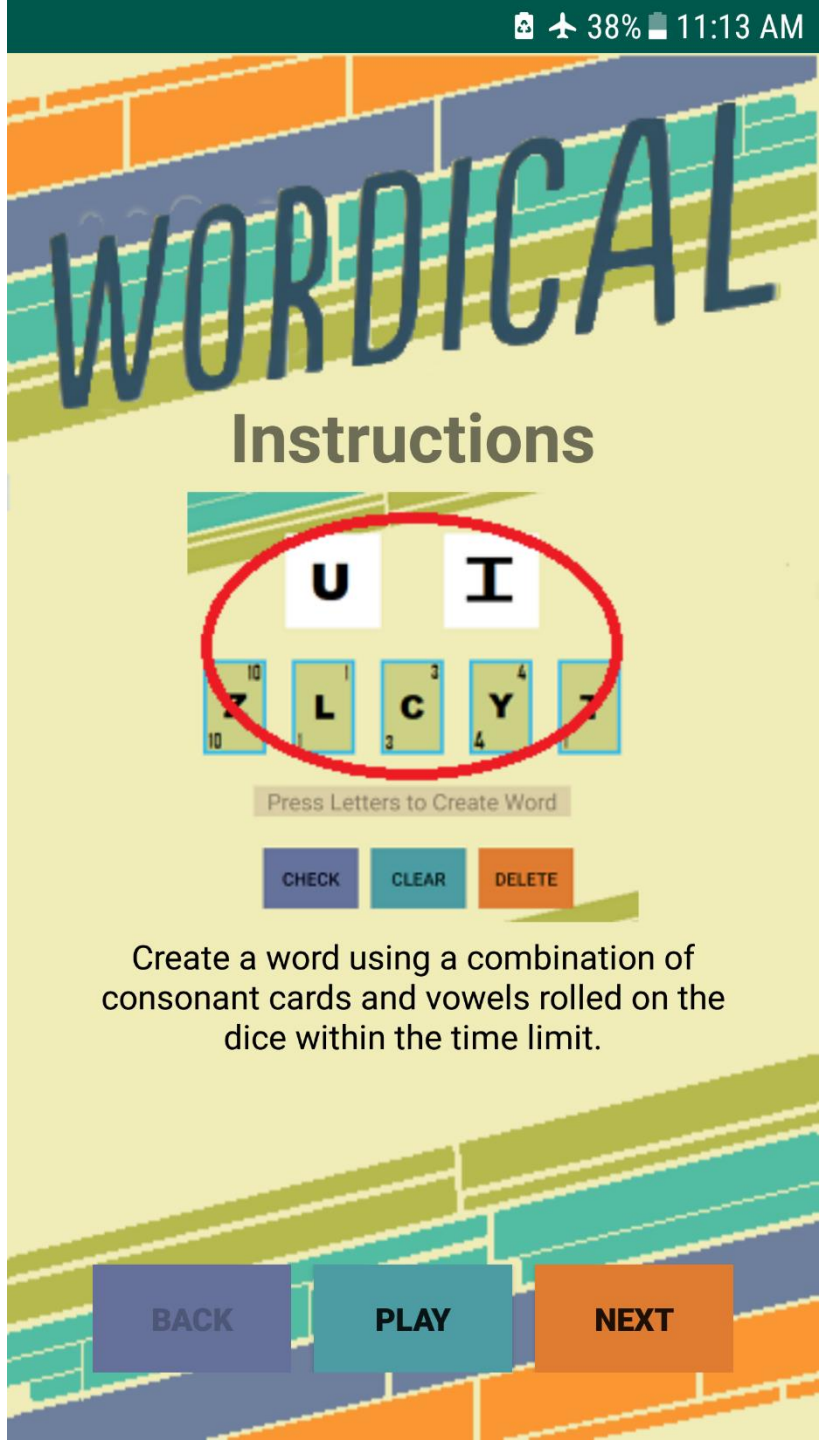
JUST JOE

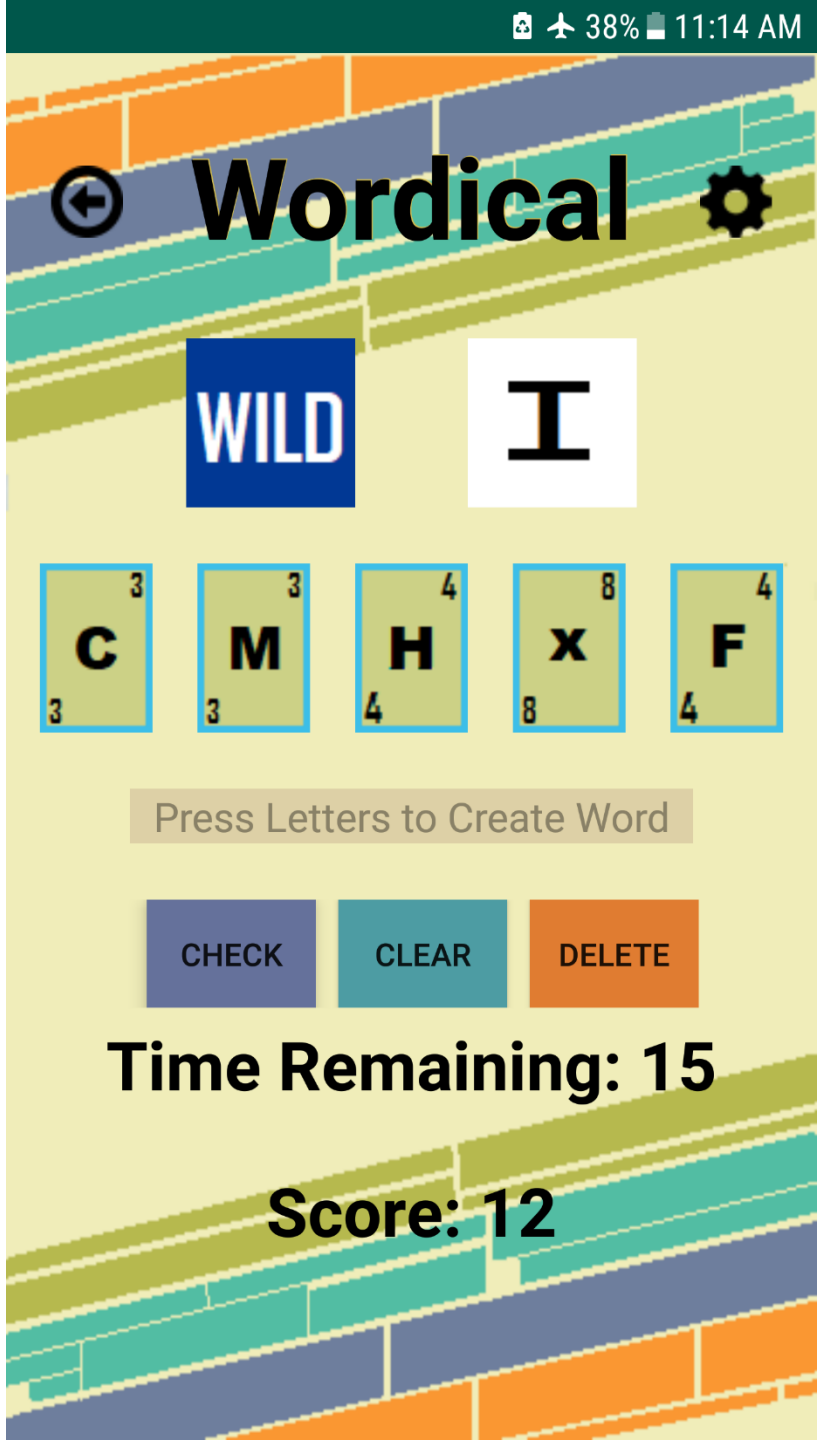
Name __ human character(s) on Sesame Street.

AMANDA : 1

NANCY : 1


WORDICAL
WILD
A
WORDICAL
THE RADICAL DICE-ROLLING, WORD-BUILDING RACE!
F4
F
F4
U
I
N1
N
N1


38% 11:13 AM
WORDICAL
Instructions
U
I
Z
L
C
Y
Press Letters to Create Word
CHECK
CLEAR
DELETE
Create a word using a combination of consonant cards and vowels rolled on the dice within the time limit.
BACK
PLAY
NEXT


38% 11:14 AM
Wordical
WILD
I
C
M
H
X
F
Press Letters to Create Word
CHECK
CLEAR
DELETE
Time Remaining: 15
Score: 12

Looking at the sheet:

1. These three classes simulate a series of addition questions on a stack of flash cards.

   Fill in the MathQuestion – the class to hold each flash card. Adapt the Stack class to hold a Stack of MathQuestions. Fill in the Runner class to add three questions to the stack.

```
public class MathQuestion {
	private int n1;
	private int n2;

	public MathQuestion(){
		n1 = (int)(Math.random()*5);
		n2 = (int)(Math.random()*5);
	}
	public MathQuestion(int range){
		n1 = (int)(Math.random()*range);
		n2 = (int)(Math.random()*range);
	}
	public String getQuestion(){
	//returns question in format 3 + 4 = ?



	}
	public int getAnswer(){
		return (n1+n2);
	}
```

```
	public void newQuestion(){
		n1 = (int)(Math.random()*5);
		n2 = (int)(Math.random()*5);
	}
	public void newQuestion(int range){



	}
	public boolean checkAnswer (int guess){
	//return true if guess equals the answer
	//false otherwise



	}
```

1. These three classes simulate a series of addition questions on a stack of flash cards.

   Fill in the MathQuestion – the class to hold each flash card. Adapt the Stack class to hold a Stack of MathQuestions. Fill in the Runner class to add three questions to the stack.

```
public class MathQuestion {
	private int n1;
	private int n2;

	public MathQuestion(){
		n1 = (int)(Math.random()*5);
		n2 = (int)(Math.random()*5);
	}
	public MathQuestion(int range){
		n1 = (int)(Math.random()*range);
		n2 = (int)(Math.random()*range);
	}
	public String getQuestion(){
	//returns question in format 3 + 4 = ?

	  return n1 + "+" + n2+ "=?";


	}
	public int getAnswer(){
		return (n1+n2);
	}
```

```
public void newQuestion(){
	n1 = (int)(Math.random()*5);
	n2 = (int)(Math.random()*5);
}
public void newQuestion(int range){



}
public boolean checkAnswer (int guess){
//return true if guess equals the answer
//false otherwise



}
```

1. These three classes simulate a series of addition questions on a stack of flash cards.

   Fill in the MathQuestion – the class to hold each flash card. Adapt the Stack class to hold a Stack of MathQuestions. Fill in the Runner class to add three questions to the stack.

```
public class MathQuestion {
	private int n1;
	private int n2;

	public MathQuestion(){
		n1 = (int)(Math.random()*5);
		n2 = (int)(Math.random()*5);
	}
	public MathQuestion(int range){
		n1 = (int)(Math.random()*range);
		n2 = (int)(Math.random()*range);
	}
	public String getQuestion(){
	//returns question in format 3 + 4 = ?

	  return n1 + "+" + n2+ "=?";


	}
	public int getAnswer(){
		return (n1+n2);
	}
```

```
public void newQuestion(){
	n1 = (int)(Math.random()*5);
	n2 = (int)(Math.random()*5);
}
public void newQuestion(int range){
	n1 = (int) (Math.random()*range;
	n2 = (int) (Math.random()*range;
}
public boolean checkAnswer (int guess){
//return true if guess equals the answer
//false otherwise



}
```

1. These three classes simulate a series of addition questions on a stack of flash cards.

   Fill in the MathQuestion – the class to hold each flash card. Adapt the Stack class to hold a Stack of MathQuestions. Fill in the Runner class to add three questions to the stack.

```
public class MathQuestion {
	private int n1;
	private int n2;

	public MathQuestion(){
		n1 = (int)(Math.random()*5);
		n2 = (int)(Math.random()*5);
	}
	public MathQuestion(int range){
		n1 = (int)(Math.random()*range);
		n2 = (int)(Math.random()*range);
	}
	public String getQuestion(){
	//returns question in format 3 + 4 = ?

	  return n1 + "+" + n2+ "=?";

	}
	public int getAnswer(){
		return (n1+n2);
	}
```

```
public void newQuestion(){
	n1 = (int)(Math.random()*5);
	n2 = (int)(Math.random()*5);
}
public void newQuestion(int range){
	n1 = (int) (Math.random()*range;
	n2 = (int) (Math.random()*range;
}
public boolean checkAnswer (int guess){
//return true if guess equals the answer
//false otherwise
	if ((n1+n2)==guess)
	  return true;
	else
	  return false;
}
```

```
public class Stack {

private int count;

private Object data[] = new Object [50];

    public Stack () {
        count = 0;
    }

    public void push (Object addMe)  {
        data [count] = addMe;
        count++;
    }

    public int size ()  {
        return count;
    }

    public boolean isFull ()  {
        return (count == 50);
    }
```

MathQuestion

```
public class MathQuestion {
        private int n1;
        private int n2;
```

```
    public Object pop () {{
        count--;
        return data [count];
    }

    public Object peek () {
        return data [count--];
    }

    public boolean isEmpty () {
        return count == 0;
    }

    public void clear () {
        count = 0;
    }
}
```

Adding A MathQuestion

```
Stack s = new Stack();
MathQuestion q = new MathQuestion();
s.push(q);
//Add two more questions
```

```
public MathQuestion(){
        n1 = (int)(Math.random()*5);
        n2 = (int)(Math.random()*5);
}
public MathQuestion(int range){
        n1 = (int)(Math.random()*range);
        n2 = (int)(Math.random()*range);
}
```

```
Stack s = new Stack();
MathQuestion q = new MathQuestion();
s.push(q);
//Add two more questions
```

1 s.push(new MathQuestion(6));

Adding A MathQuestion

```
public MathQuestion(){
        n1 = (int)(Math.random()*5);
        n2 = (int)(Math.random()*5);
}
public MathQuestion(int range){
        n1 = (int)(Math.random()*range);
        n2 = (int)(Math.random()*range);
}
```

```
Stack s = new Stack();
MathQuestion q = new MathQuestion();
s.push(q);
//Add two more questions
```

1. s.push(new MathQuestion(6));
2. s.push(new MathQuestion());

Adding A MathQuestion

```
public MathQuestion(){
        n1 = (int)(Math.random()*5);
        n2 = (int)(Math.random()*5);
}
public MathQuestion(int range){
        n1 = (int)(Math.random()*range);
        n2 = (int)(Math.random()*range);
}
```

Adding A MathQuestion

```
Stack s = new Stack();
MathQuestion q = new MathQuestion();
s.push(q);
//Add two more questions
```

```
public MathQuestion(){
        n1 = (int)(Math.random()*5);
        n2 = (int)(Math.random()*5);
}
public MathQuestion(int range){
        n1 = (int)(Math.random()*range);
        n2 = (int)(Math.random()*range);
}
```

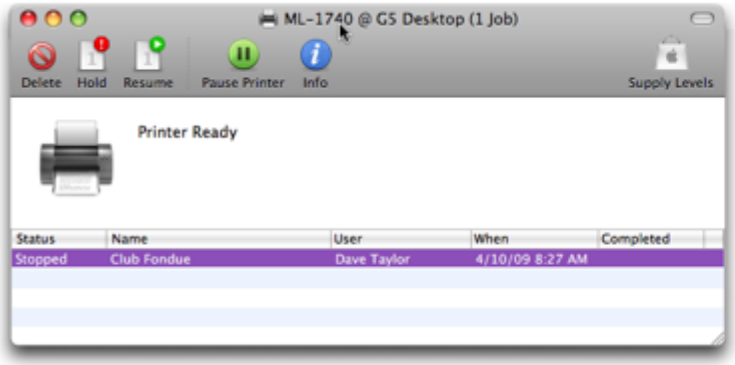
1. s.push(new MathQuestion(6));
2. s.push(new MathQuestion());
3. MathQuestion q1 = new MathQuestion(25);
   s.push(q1);

Adding A MathQuestion

```
Stack s = new Stack();
MathQuestion q = new MathQuestion();
s.push(q);
//Add two more questions
```

1. `s.push(new MathQuestion(6));`
2. `s.push(new MathQuestion());`
3.
```
MathQuestion q1 = new MathQuestion(25);
s.push(q1);
```
4.
```
MathQuestion q2 = new MathQuestion();
s.push(q2);
```

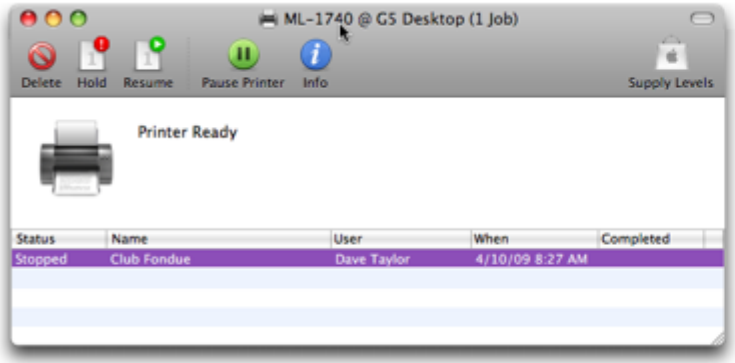
```
public MathQuestion(){
        n1 = (int)(Math.random()*5);
        n2 = (int)(Math.random()*5);
}
public MathQuestion(int range){
        n1 = (int)(Math.random()*range);
        n2 = (int)(Math.random()*range);
}
```



2. This code simulates a printer queue.
   (a) Fill in the PrinterJob Object
   (b) Adapt the Queue class so that it stores a queue of PrintJobs.
   (c) Create a Queue. Add two PrintJobs to the Queue.

```
public class PrinterJob {

    private String filename;
    private double time;

    public PrinterJob(){



    }
    public PrinterJob(String fn, double t){



    }
    public void setFileName(String fn){

    }
    public void setTime(double t){

    }
    public String getFileName(){

    }
```

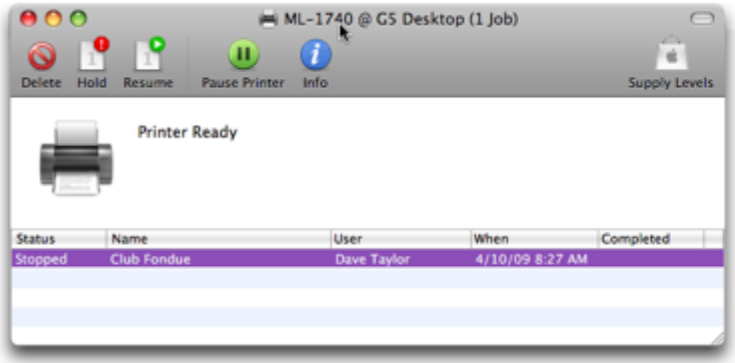
```
public double getTime(){

}
public boolean equals(PrinterJob j){




}
public int compareTo(PrinterJob j){
    if(time < ____________________)
        return -1;
    else if(time == ____________________)
        return 0;
    else
        return 1;
}
public String toString(){
//returns a phrase like Temp1.doc was
//  added to the printer queue at 3.30




}
```



2. This code simulates a printer queue.
   (a) Fill in the PrinterJob Object
   (b) Adapt the Queue class so that it stores a queue of PrintJobs.
   (c) Create a Queue. Add two PrintJobs to the Queue.

```
public class PrinterJob {

    private String filename;
    private double time;

    public PrinterJob(){
        filename = "name.doc";
        time = 6.30;
    }
    public PrinterJob(String fn, double t){
        filename = fn;
        time = t;
    }
    public void setFileName(String fn){

    }
    public void setTime(double t){

    }
    public String getFileName(){

    }
```

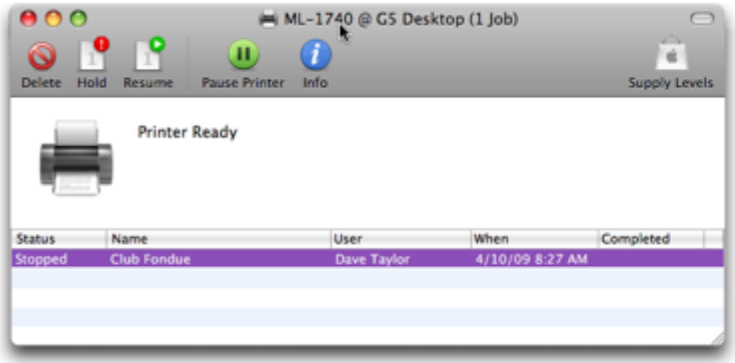
```
public double getTime(){

}
public boolean equals(PrinterJob j){




}
public int compareTo(PrinterJob j){
    if(time < ________________)
        return -1;
    else if(time == ________________)
        return 0;
    else
        return 1;
}
public String toString(){
//returns a phrase like Temp1.doc was
//  added to the printer queue at 3.30




}
```



2. This code simulates a printer queue.
   (a) Fill in the PrinterJob Object
   (b) Adapt the Queue class so that it stores a queue of PrintJobs.
   (c) Create a Queue. Add two PrintJobs to the Queue.

```
public class PrinterJob {

    private String filename;
    private double time;

    public PrinterJob(){
        filename = "name.doc";
        time = 6.30;
    }
    public PrinterJob(String fn, double t){
        filename = fn;
        time = t;
    }
    public void setFileName(String fn){
        filename = fn;
    }
    public void setTime(double t){
         time = t;
    }
    public String getFileName(){
         return filename;
    }
```

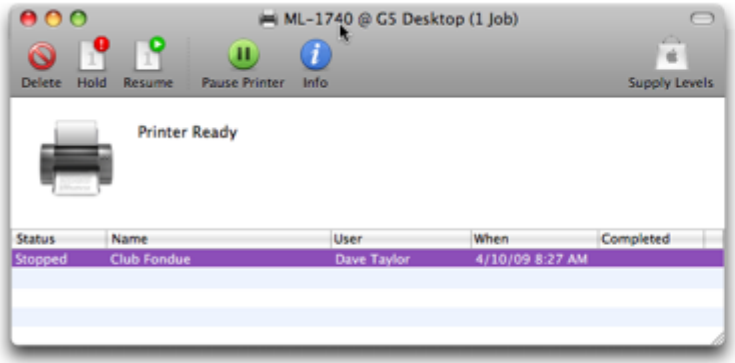
```
public double getTime(){

}
public boolean equals(PrinterJob j){




}
public int compareTo(PrinterJob j){
    if(time < ________________)
        return -1;
    else if(time == ________________)
        return 0;
    else
        return 1;
}
public String toString(){
//returns a phrase like Temp1.doc was
//  added to the printer queue at 3.30




}
```



2. This code simulates a printer queue.
   (a) Fill in the PrinterJob Object
   (b) Adapt the Queue class so that it stores a queue of PrintJobs.
   (c) Create a Queue. Add two PrintJobs to the Queue.

```
public class PrinterJob {

    private String filename;
    private double time;

    public PrinterJob(){
        filename = "name.doc";
        time = 6.30;
    }
    public PrinterJob(String fn, double t){
        filename = fn;
        time = t;
    }
    public void setFileName(String fn){
        filename = fn;
    }
    public void setTime(double t){
         time = t;
    }
    public String getFileName(){
         return filename;
    }
```

```
public double getTime(){
     return time;
}
public boolean equals(PrinterJob j){
  return j.getTime()==time &&
    j.getFileName().equals(filename))


}
public int compareTo(PrinterJob j){
    if(time < _________________)
        return -1;
    else if(time == _________________)
        return 0;
    else
        return 1;
}
public String toString(){
//returns a phrase like Temp1.doc was
//  added to the printer queue at 3.30




}
```



2. This code simulates a printer queue.
   (a) Fill in the PrinterJob Object
   (b) Adapt the Queue class so that it stores a queue of PrintJobs.
   (c) Create a Queue. Add two PrintJobs to the Queue.

```
public class PrinterJob {

    private String filename;
    private double time;

    public PrinterJob(){
        filename = "name.doc";
        time = 6.30;
    }
    public PrinterJob(String fn, double t){
        filename = fn;
        time = t;
    }
    public void setFileName(String fn){
        filename = fn;
    }
    public void setTime(double t){
        time = t;
    }
    public String getFileName(){
        return filename;
    }
```

```
public double getTime(){
    return time;
}
public boolean equals(PrinterJob j){
  return j.getTime()==time &&
    j.getFileName().equals(filename))


}
public int compareTo(PrinterJob j){
    if(time < ____j.getTime()__)
        return -1;
    else if(time == __j.getTime()___)
        return 0;
    else
        return 1;
}
public String toString(){
//returns a phrase like Temp1.doc was
//  added to the printer queue at 3.30
  return filename + "was added
    to the printer queue at" +time;

}
```

PrinterJob

```
public class Queue {

  private Object data[] = new Object [50];
  private int count;
  private int head;

  public Queue () {
    count = 0;
    head = 0;
  }

  public void enqueue (Object value) {
    int tail = (head + count) % data.length;
    data [tail] = value;
    count++;
  }

  public Object dequeue () {
    Object temp = data [head];
    count--;
    head = (head + 1) % data.length;
    return temp;
  }
```

```
public class PrinterJob {

    private String filename;
    private double time;
```

```
  public Object peek () {
    return data [head];
  }

  public int size () {
    return count;
  }

  public boolean isEmpty () {
    return (count == 0);
  }

}
```

Create a Queue.
Add two Print Jobs to the queue.

Queue q = new Queue();

1. q.enqueue(new PrinterJob());
2. q.enqueue(new PrinterJob("hello.png", 3:30);
3. PrinterJob p = new PrinterJob("frog.pdf", 5:00);
   q.enqueue(p);
4. PrinterJob p2 = new PrinterJob();
   q.enqueue(p2);

```
public class PrinterJob {

    private String filename;
    private double time;

    public PrinterJob(){

    }
    public PrinterJob(String fn, double t){

    }
```

```
public class Queue {

  private Object data[] = new Object [50];
  private int count;
  private int head;

  public Queue () {
    count = 0;
    head = 0;
  }
```

Pictures/ Rough Files