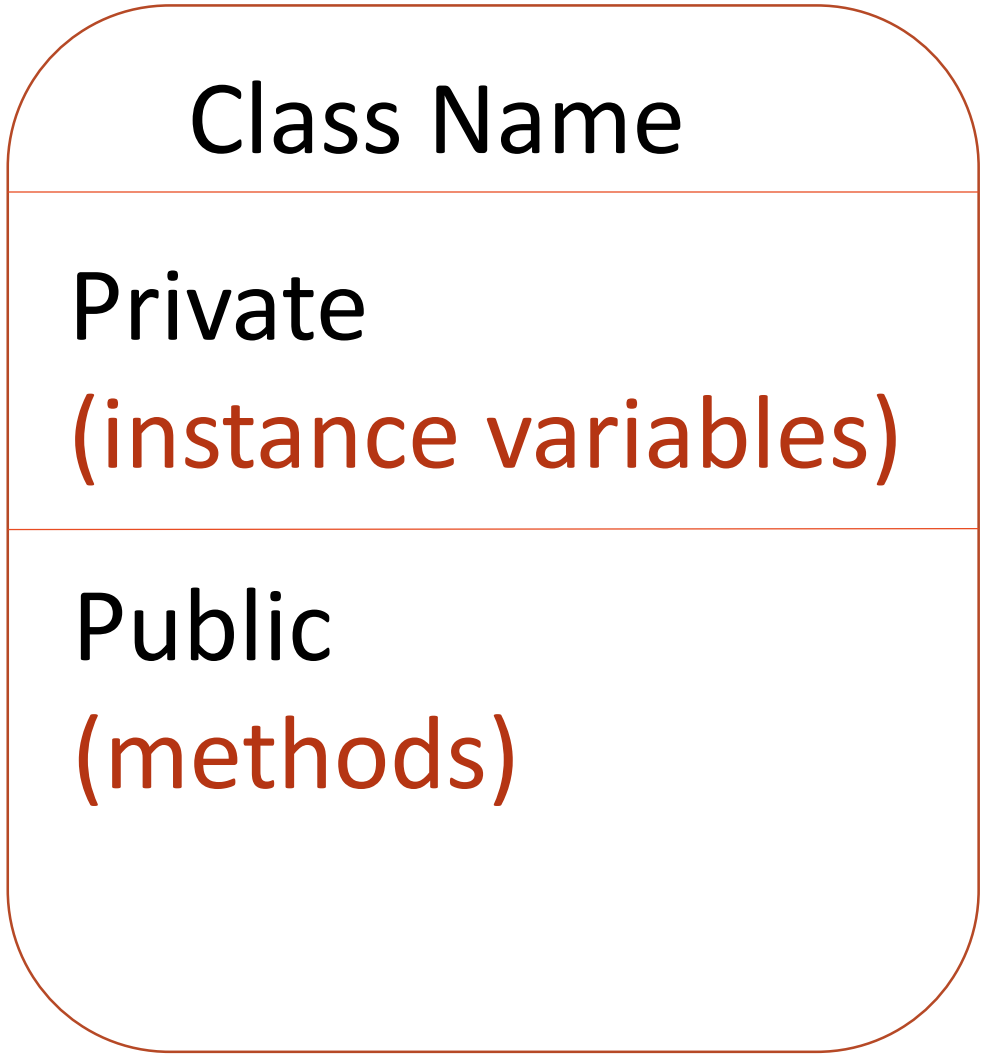
UML

Unified Modelling Language
Object Diagrams



A UML is a diagram that is used to design classes and objects.

It is divided into three pieces, as shown.

Recall our card object from yesterday:

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

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

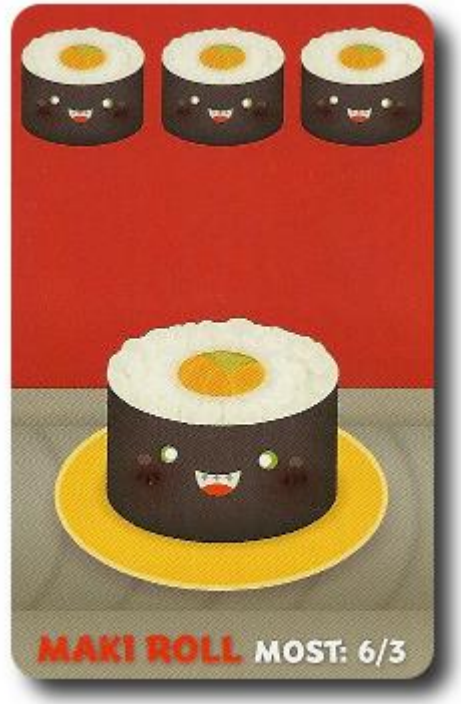
Facilitator

Sort by name, then by num

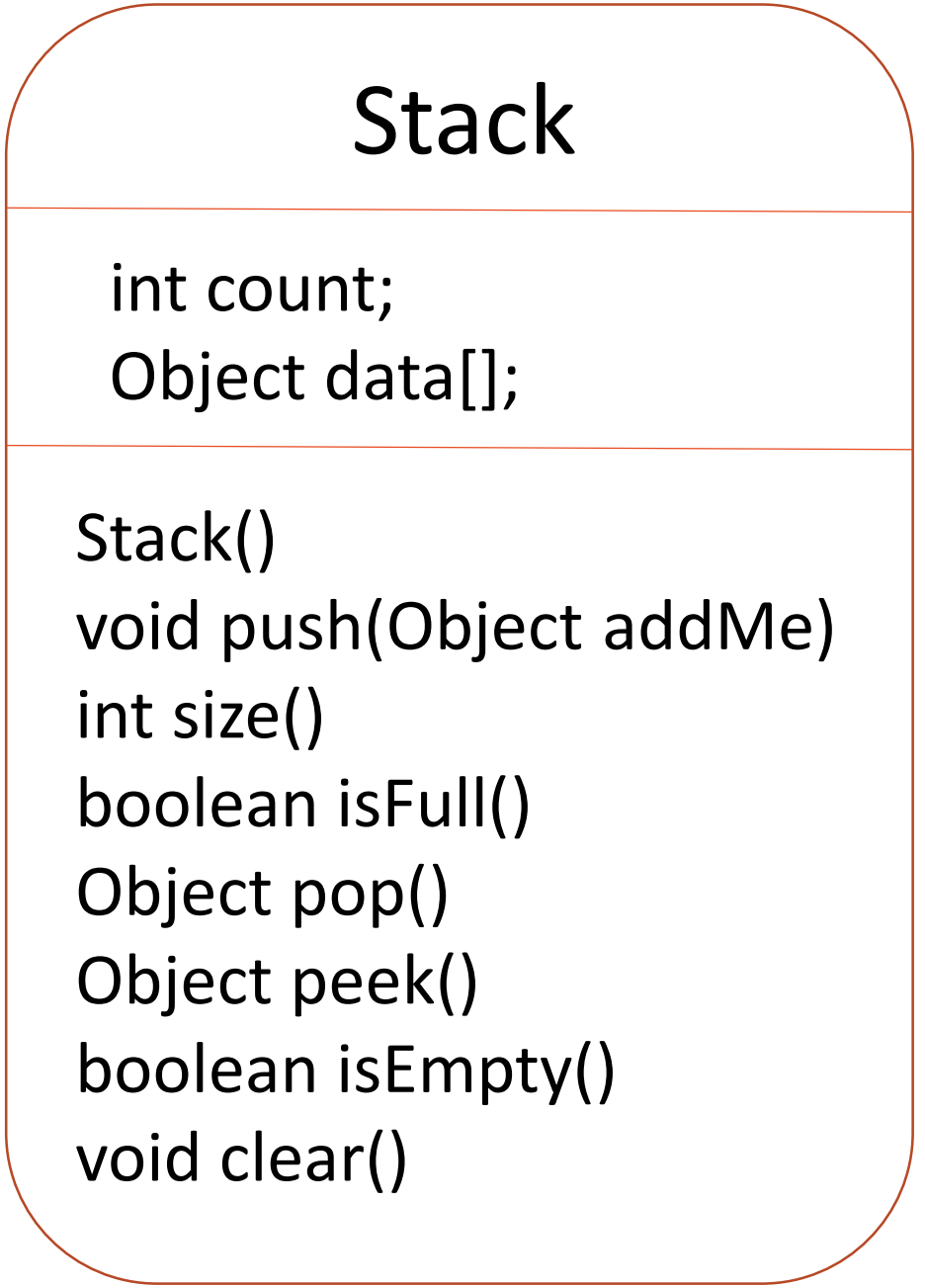
Sushi

int num;
String name;

Sushi()
Sushi(int n, String na)
String getName()
int getNum()
int getPic()
String getMsg()
boolean equals(Sushi s)
int compareTo(Sushi s)



```
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 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

Sushi

int num;
String name;

Sushi()
Sushi(int n, String na)
String getName()
int getNum()
int getPic()
String getMsg()
boolean equals(Sushi s)
int compareTo(Sushi s)

SushiStack

int count;
Sushi data[];

SushiStack()
void push(Sushi addMe)
int size()
boolean isFull()
Sushi pop()
Sushi peek()
boolean isEmpty()
void clear()

SushiStackRunner

SushiStack s[];

Sushi

int num;
String name;

Sushi()
Sushi(int n, String na)
String getName()
int getNum()
int getPic()
String getMsg()
boolean equals(Sushi s)
int compareTo(Sushi s)

SushiStack

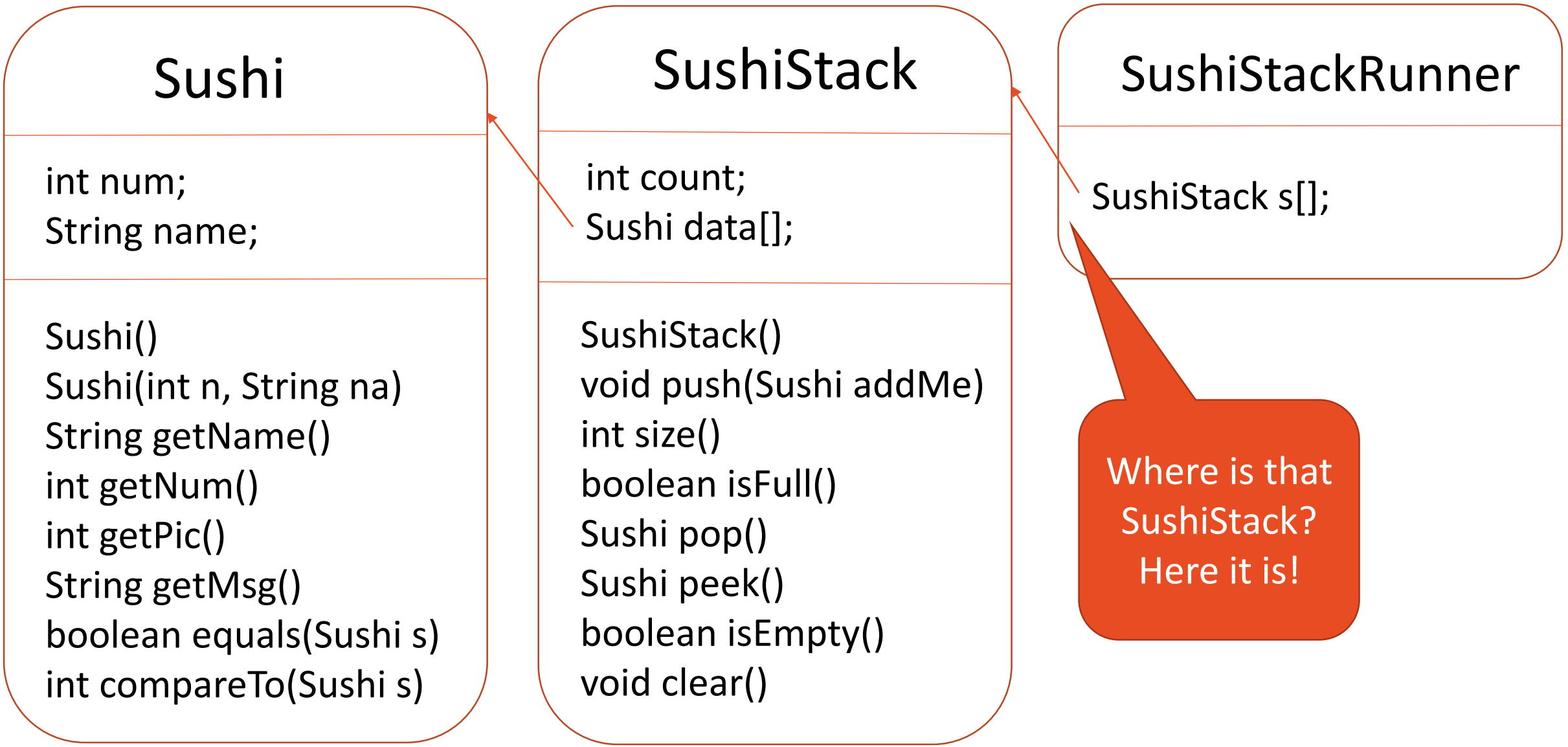
int count;
Sushi data[];

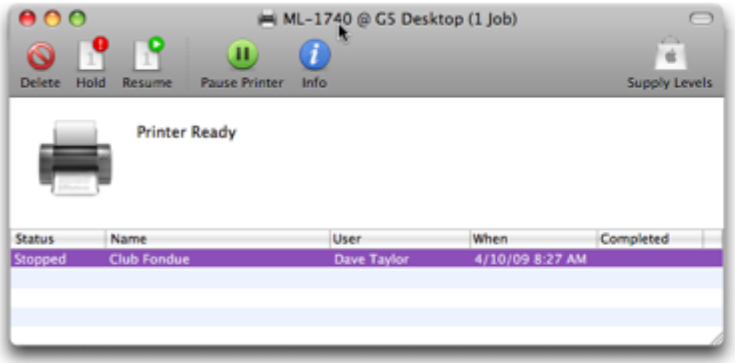
SushiStack()
void push(Sushi addMe)
int size()
boolean isFull()
Sushi pop()
Sushi peek()
boolean isEmpty()
void clear()

SushiStackRunner

SushiStack s[];

We indicate dependence using an arrow.

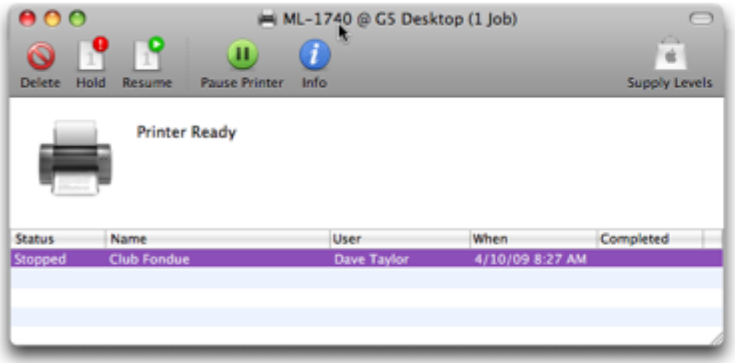

Sushi
int num;
String name;
Sushi()
Sushi(int n, String na)
String getName()
int getNum()
int getPic()
String getMsg()
boolean equals(Sushi s)
int compareTo(Sushi s)
SushiStack
int count;
Sushi data[];
SushiStack()
void push(Sushi addMe)
int size()
boolean isFull()
Sushi pop()
Sushi peek()
boolean isEmpty()
void clear()
SushiStackRunner
SushiStack s[];
Where is that SushiStack? Here it is!



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;

}
```

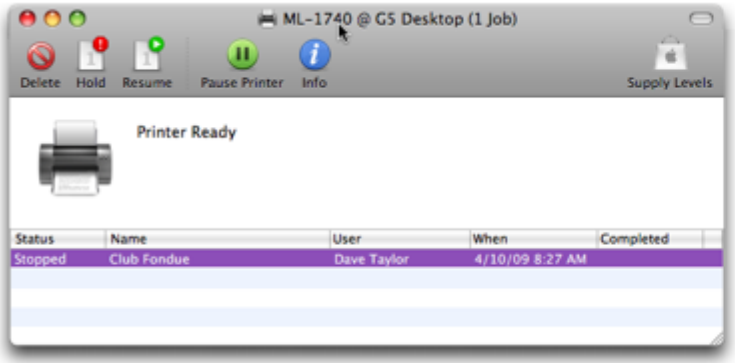


2. This code simulates a printer queue.
   (a) Fill in the PrinterJob Object
   (b) Adapt the Queue class so that it stores a queue of PrintJobs.
   (c) Create a Queue. Add two PrintJobs to the Queue.

```
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;

}
```

Printer Job
String filename; double time;



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;

}
```

Printer Job
String filename; double time;
PrinterJob () PrinterJob (String fn, double t) void setFileName (String fn) void setTime (double t) String getFileName () double getTime() boolean equals(PrinterJob c) int compareTo(PrinterJob c) String toString ()

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.

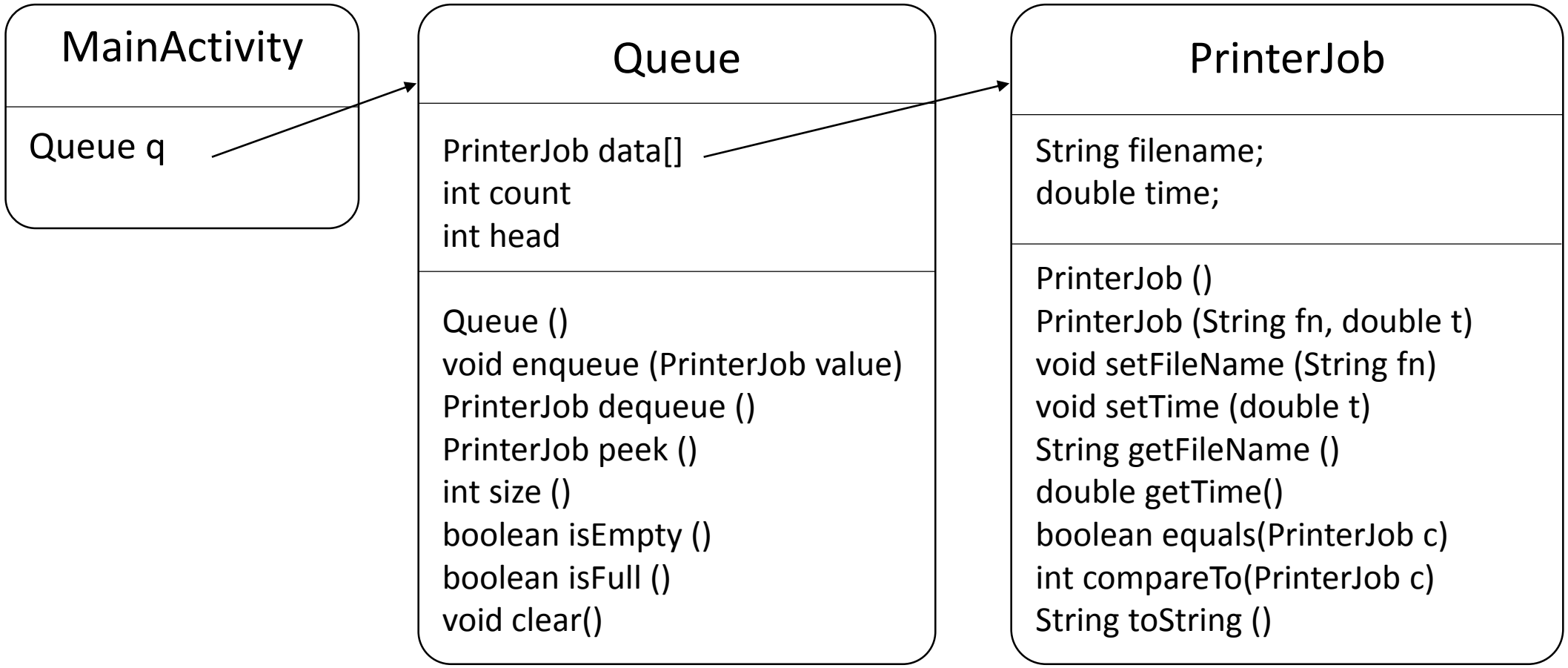
Queue q = new Queue();

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;
  }
```

The diagram for the entire program would look like this:



Declaration

Setting aside RAM

“JButton b;”

Constructor

Sets up dynamic memory

to be ready for use

“new”

Facilitator

Anything that isn’t construction, accessor, mutator

Complex functions

“equals”

“compareTo”

Mutator

Changing

dynamic memory

“set” - setForeground

Accessor

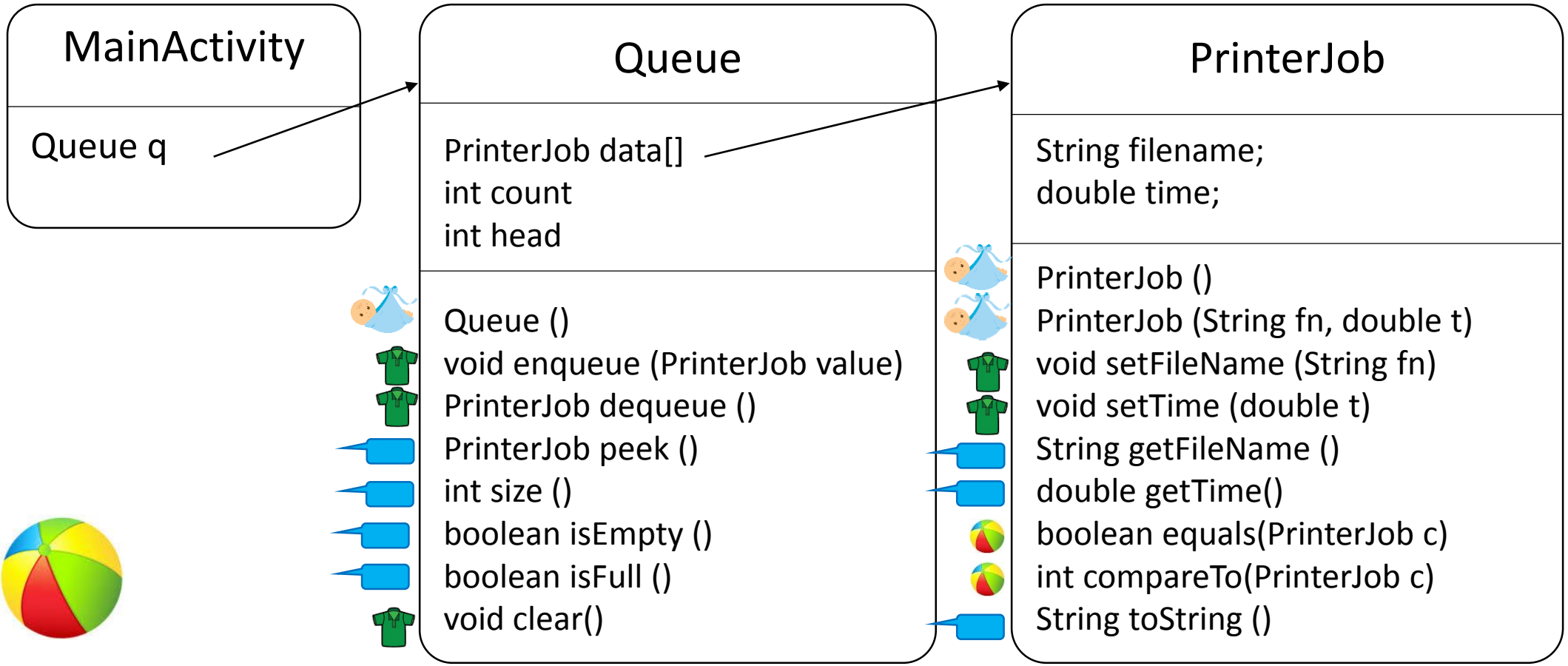
Checking dynamic memory

“get” – getText

“toString”

“is” – isVisible

We can classify the methods quickly:

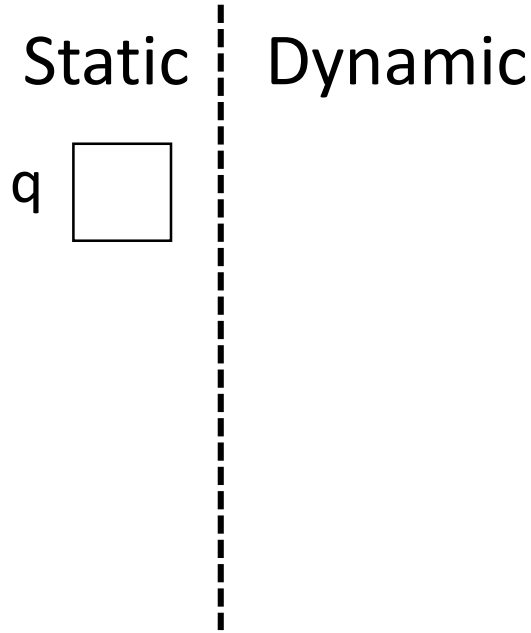


Object Memory Diagrams

Create an memory diagram for this code.

```
Queue q = new Queue();
q.enqueue(new PrinterJob());
q.enqueue(new PrinterJob("hello.png", 3:30);
q.dequeue();
PrinterJob p = new PrinterJob("bye.png", 4:30);
q.enqueue(p);
```

Memory Diagram



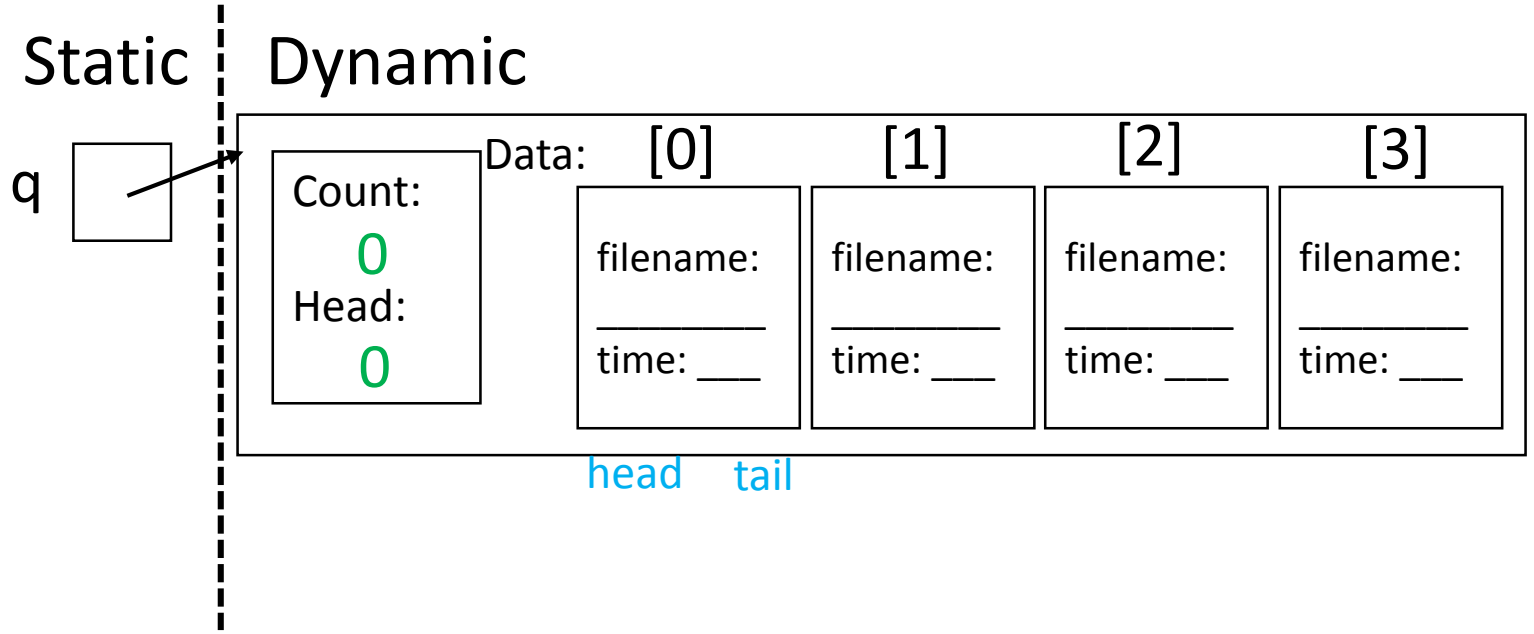

```
public class PrinterJob {

    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;
  }

  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;
  }
```

Create an memory diagram for this code.

```
Queue q = new Queue();
q.enqueue(new PrinterJob());
q.enqueue(new PrinterJob("hello.png", 3:30);
q.dequeue();
PrinterJob p = new PrinterJob("bye.png", 4:30);
q.enqueue(p);
```

Memory Diagram



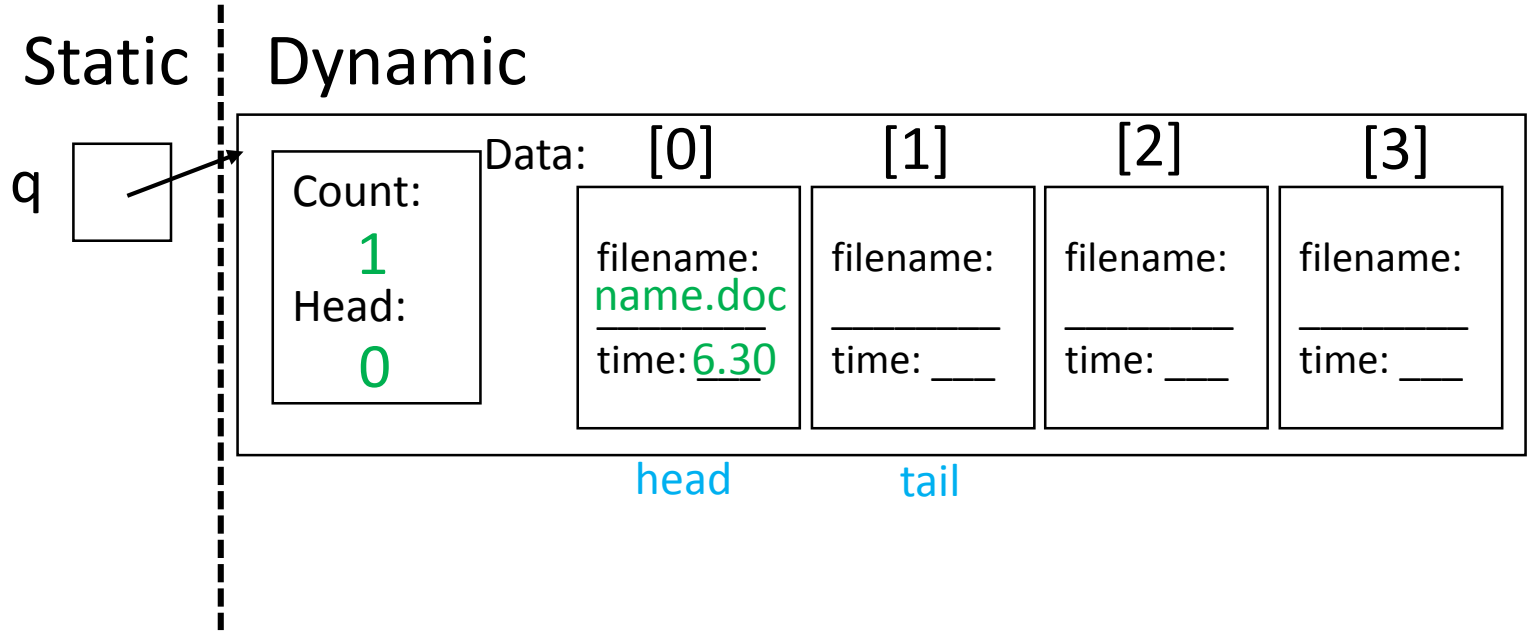

```
public class PrinterJob {

    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;
  }

  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;
  }
```

Create an memory diagram for this code.

```
Queue q = new Queue();
q.enqueue(new PrinterJob());
q.enqueue(new PrinterJob("hello.png", 3:30);
q.dequeue();
PrinterJob p = new PrinterJob("bye.png", 4:30);
q.enqueue(p);
```

Memory Diagram



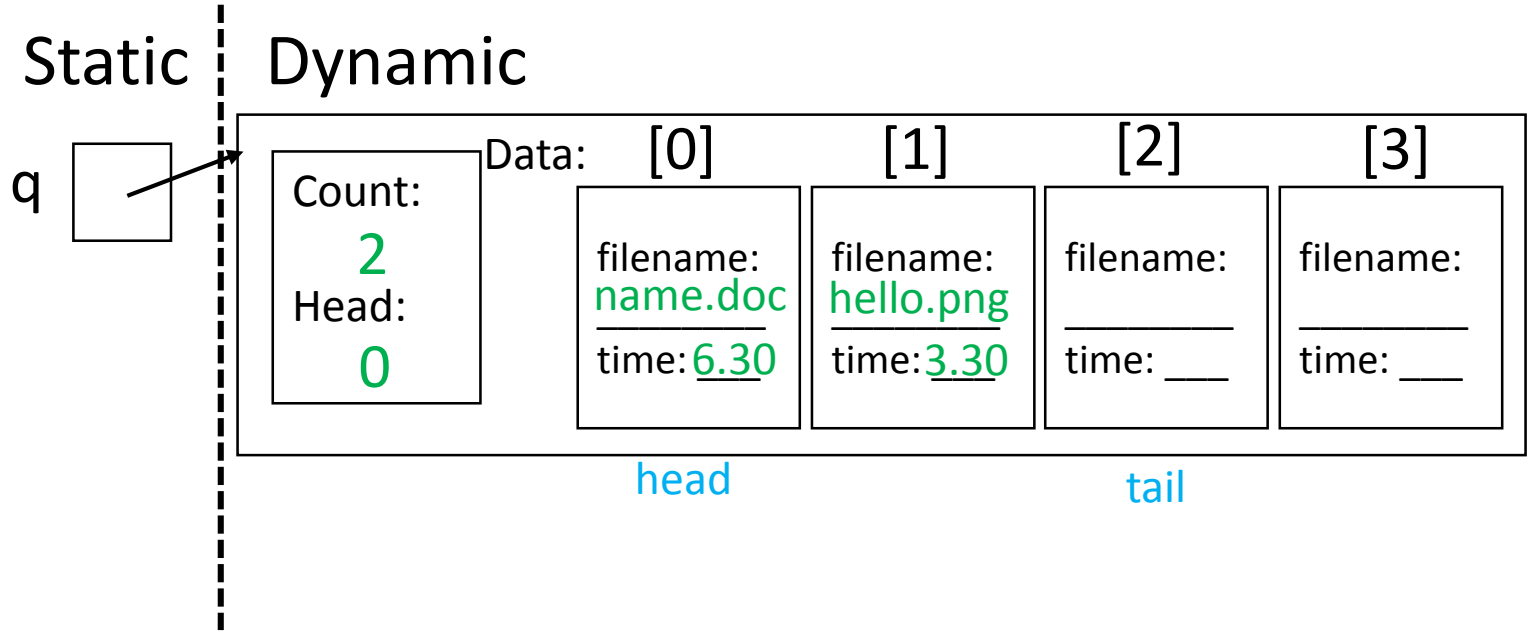

```
public class PrinterJob {

    private String filename;
    private double time;

    public PrinterJob(){
       filename = "name.doc";  time = 6.30;
    }
    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;
  }

  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;
  }
```

Create an memory diagram for this code.

```
Queue q = new Queue();
q.enqueue(new PrinterJob());
q.enqueue(new PrinterJob("hello.png", 3:30);
q.dequeue();
PrinterJob p = new PrinterJob("bye.png", 4:30);
q.enqueue(p);
```

Memory Diagram



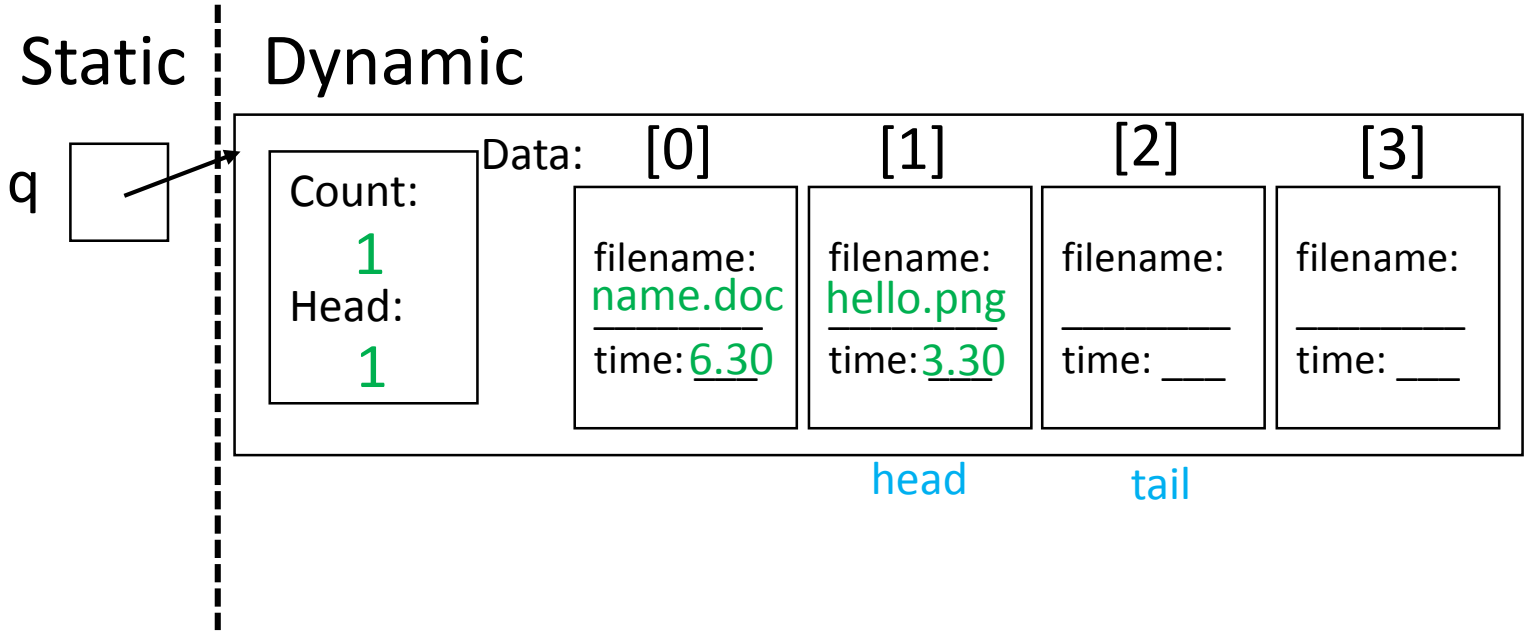

```
public class PrinterJob {

   private String filename;
   private double time;

   public PrinterJob(){
      filename = "name.doc";  time = 6.30;
   }
   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;
  }

  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;
  }
```

Create an memory diagram for this code.

```
Queue q = new Queue();
q.enqueue(new PrinterJob());
q.enqueue(new PrinterJob("hello.png", 3:30);
q.dequeue();
PrinterJob p = new PrinterJob("bye.png", 4:30);
q.enqueue(p);
```

Memory Diagram



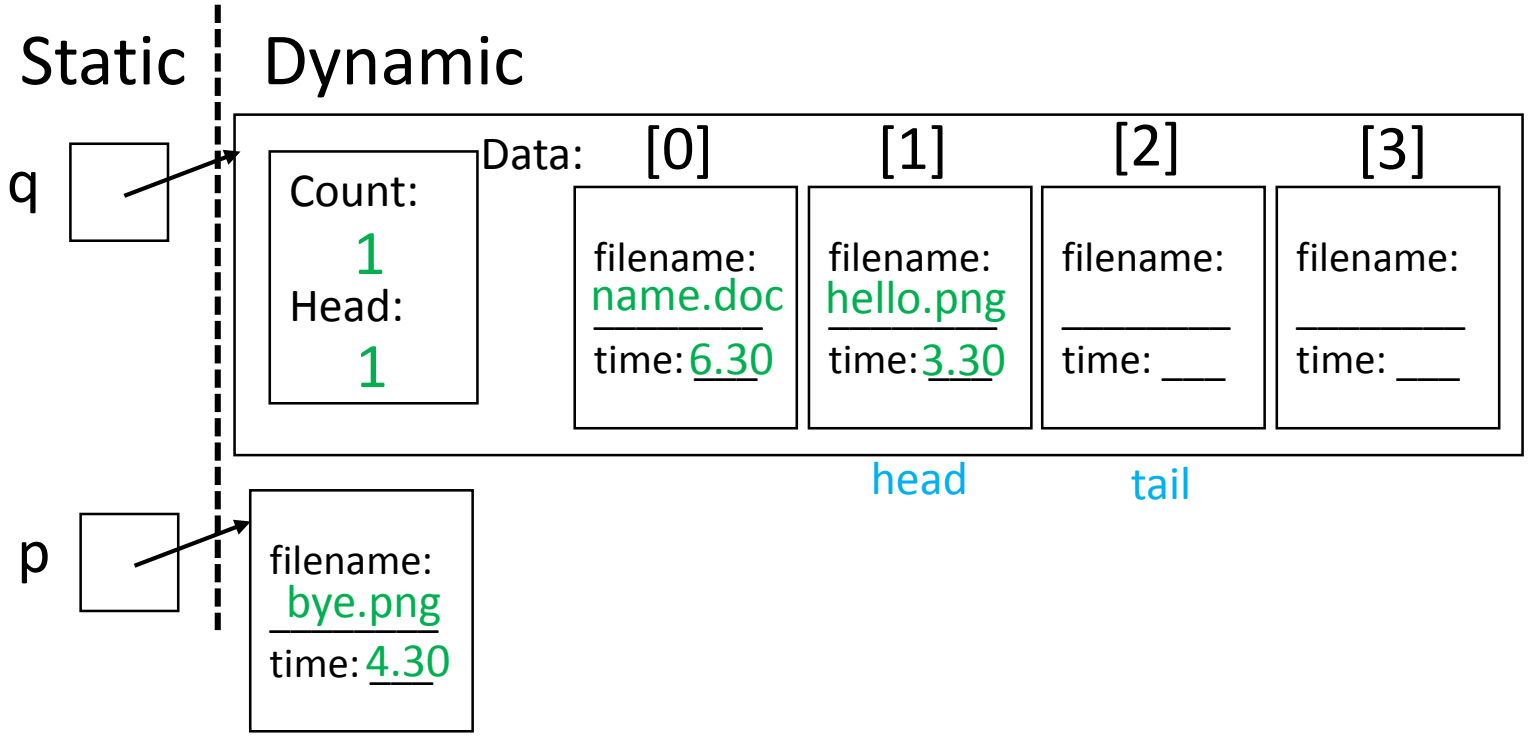

```
public class PrinterJob {

    private String filename;
    private double time;

    public PrinterJob(){
       filename = "name.doc";  time = 6.30;
    }
    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;
  }

  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;
  }
```

Create an memory diagram for this code.

```
Queue q = new Queue();
q.enqueue(new PrinterJob());
q.enqueue(new PrinterJob("hello.png", 3:30);
q.dequeue();
PrinterJob p = new PrinterJob("bye.png", 4:30);
q.enqueue(p);
```

Memory Diagram



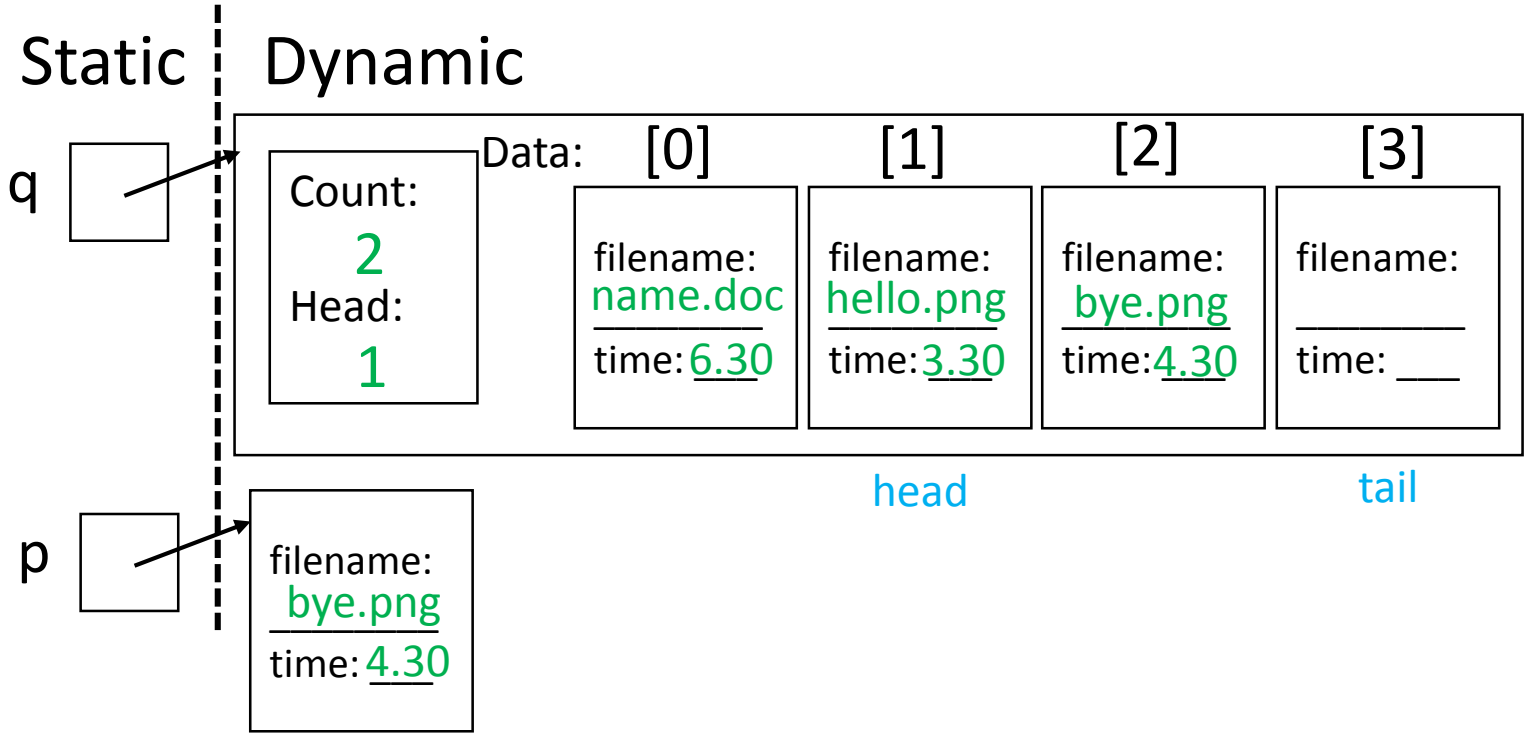

```
public class PrinterJob {

    private String filename;
    private double time;

    public PrinterJob(){
       filename = "name.doc";  time = 6.30;
    }
    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;
  }

  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;
  }
```

Create an memory diagram for this code.

```
Queue q = new Queue();
q.enqueue(new PrinterJob());
q.enqueue(new PrinterJob("hello.png", 3:30);
q.dequeue();
PrinterJob p = new PrinterJob("bye.png", 4:30);
q.enqueue(p);
```

Memory Diagram



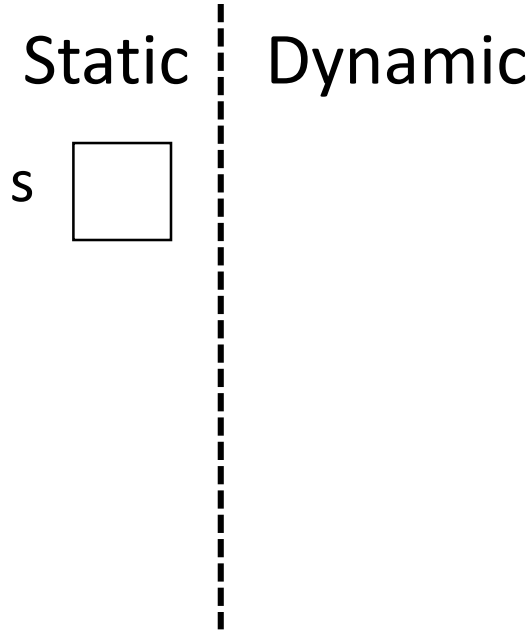

```
public class PrinterJob {

    private String filename;
    private double time;

    public PrinterJob(){
       filename = "name.doc";  time = 6.30;
    }
    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;
  }

  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;
  }
```

Create an memory diagram for this code.

```
SushiStack s = new SushiStack ();
s.push (new Sushi (1,"Egg Nigiri"));
s.push (new Sushi (6,"Spoon"));
s.push (new Sushi ());
s.pop();
s.push (new Sushi (4,"Sashimi"));
s.pop();
```

Memory Diagram




```
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;
   }
```

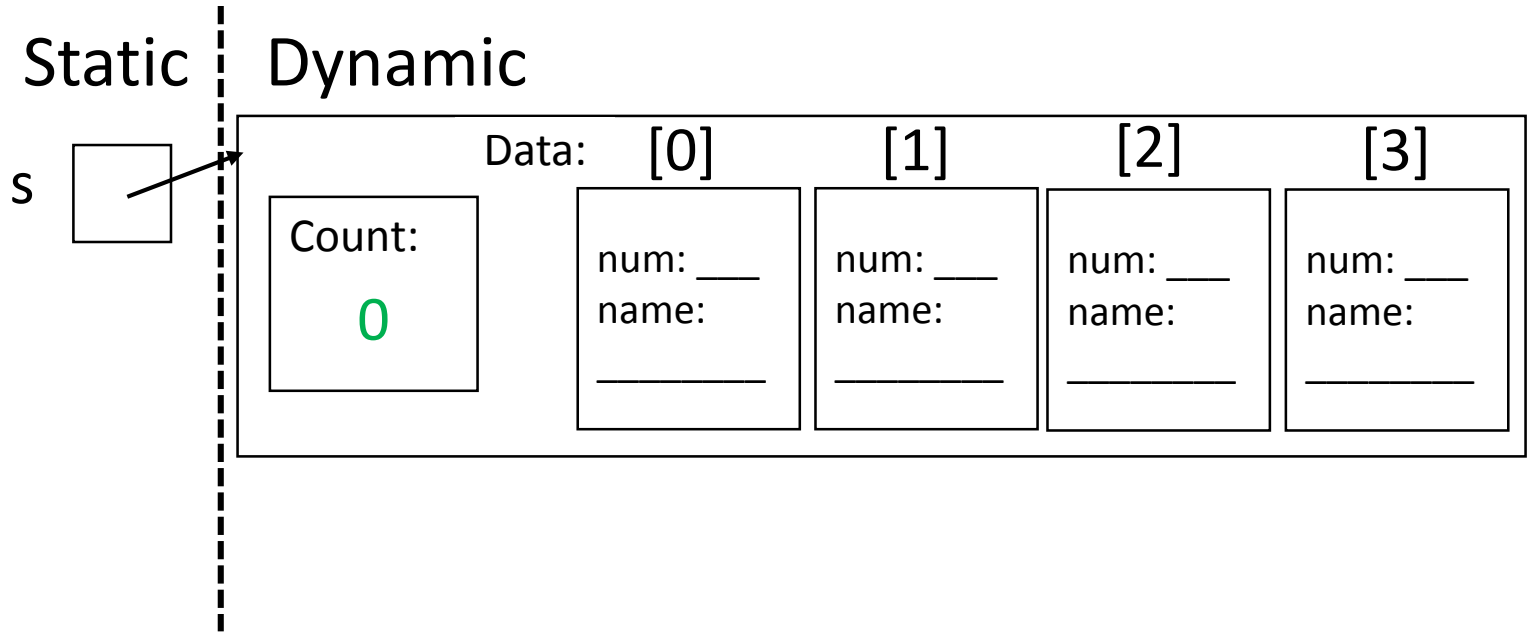
```
public class Stack {

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

public Stack () {
    count = 0;
}

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

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

Create an memory diagram for this code.

```
SushiStack s = new SushiStack ();
s.push (new Sushi (1,"Egg Nigiri"));
s.push (new Sushi (6,"Spoon"));
s.push (new Sushi ());
s.pop();
s.push (new Sushi (4,"Sashimi"));
s.pop();
```

```
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;
   }
```

Memory Diagram



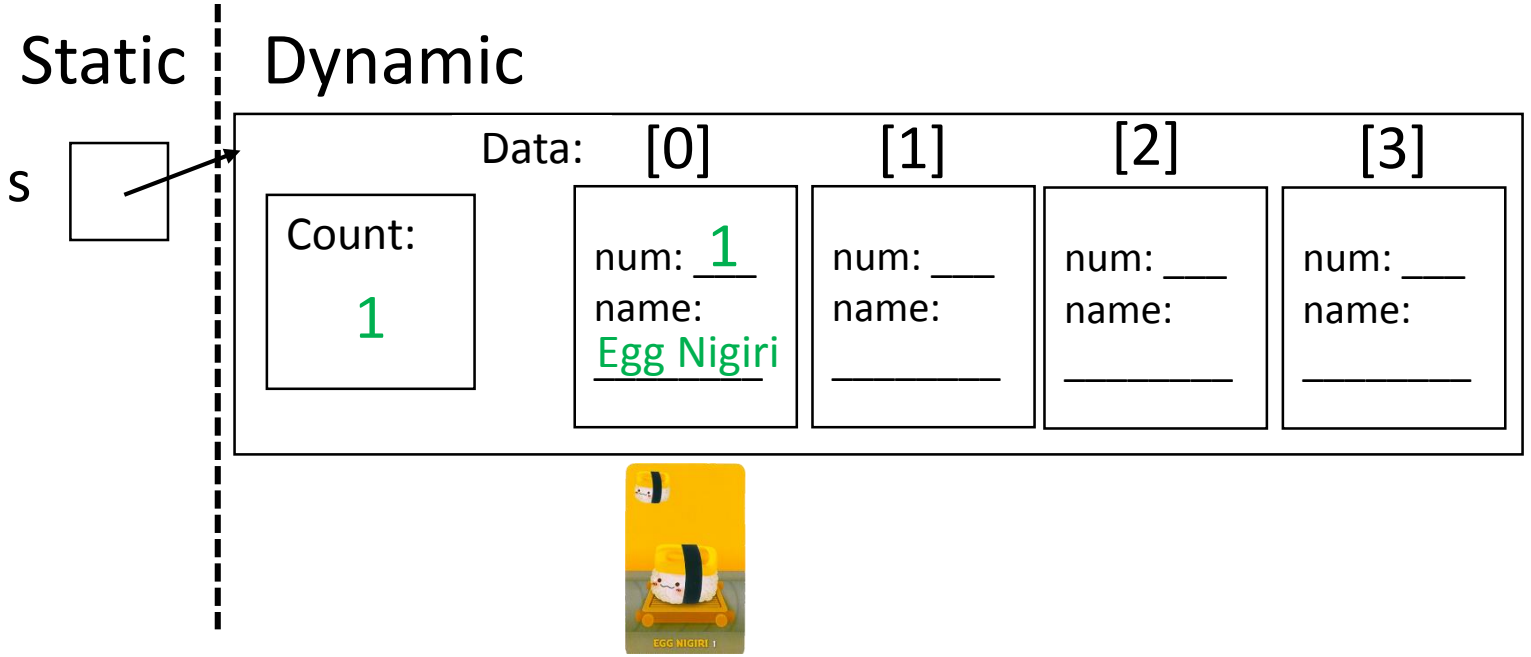

```
public class Stack {

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

public Stack () {
    count = 0;
}

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

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

Create an memory diagram for this code.

```
SushiStack s = new SushiStack ();
s.push (new Sushi (1,"Egg Nigiri"));
s.push (new Sushi (6,"Spoon"));
s.push (new Sushi ());
s.pop();
s.push (new Sushi (4,"Sashimi"));
s.pop();
```

```
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;
   }
```

Memory Diagram



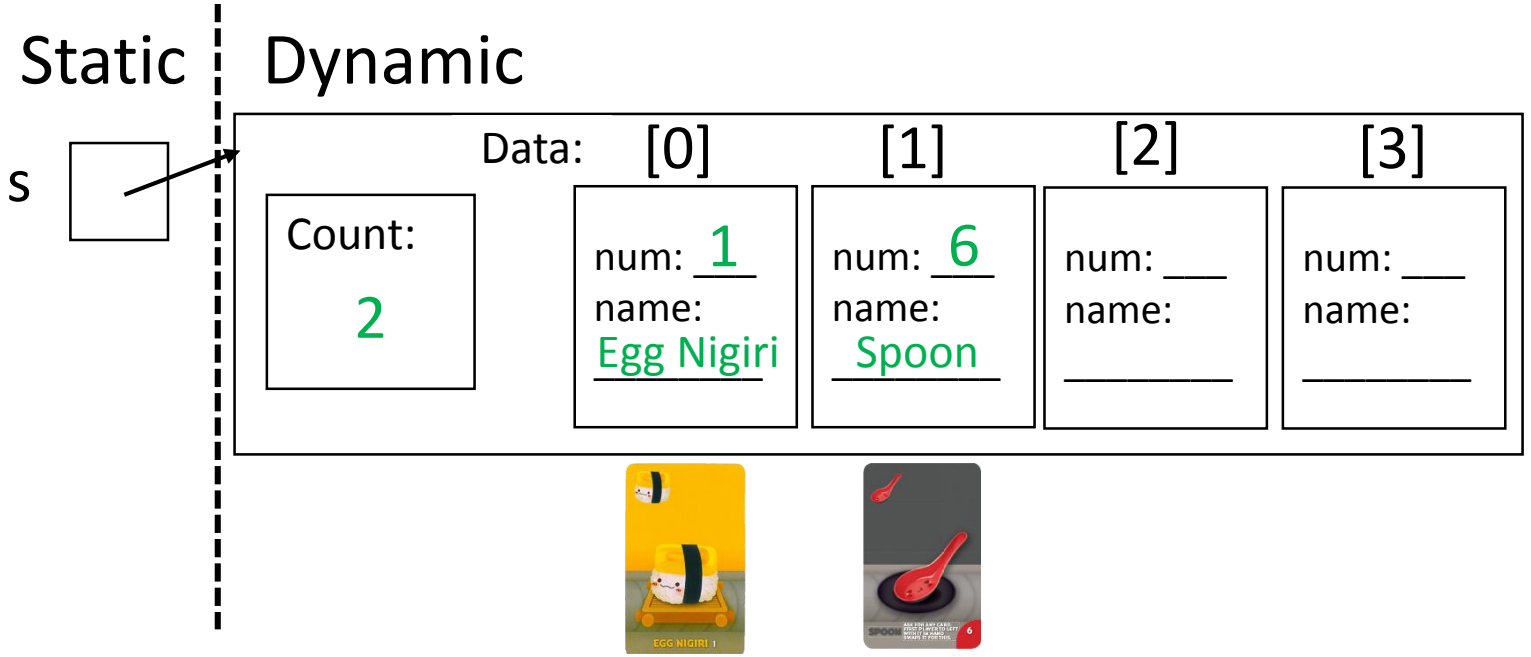

```
public class Stack {

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

public Stack () {
    count = 0;
}

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

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

Create an memory diagram for this code.

```
SushiStack s = new SushiStack ();
s.push (new Sushi (1,"Egg Nigiri"));
s.push (new Sushi (6,"Spoon"));
s.push (new Sushi ());
s.pop();
s.push (new Sushi (4,"Sashimi"));
s.pop();
```

```
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;
   }
```

Memory Diagram



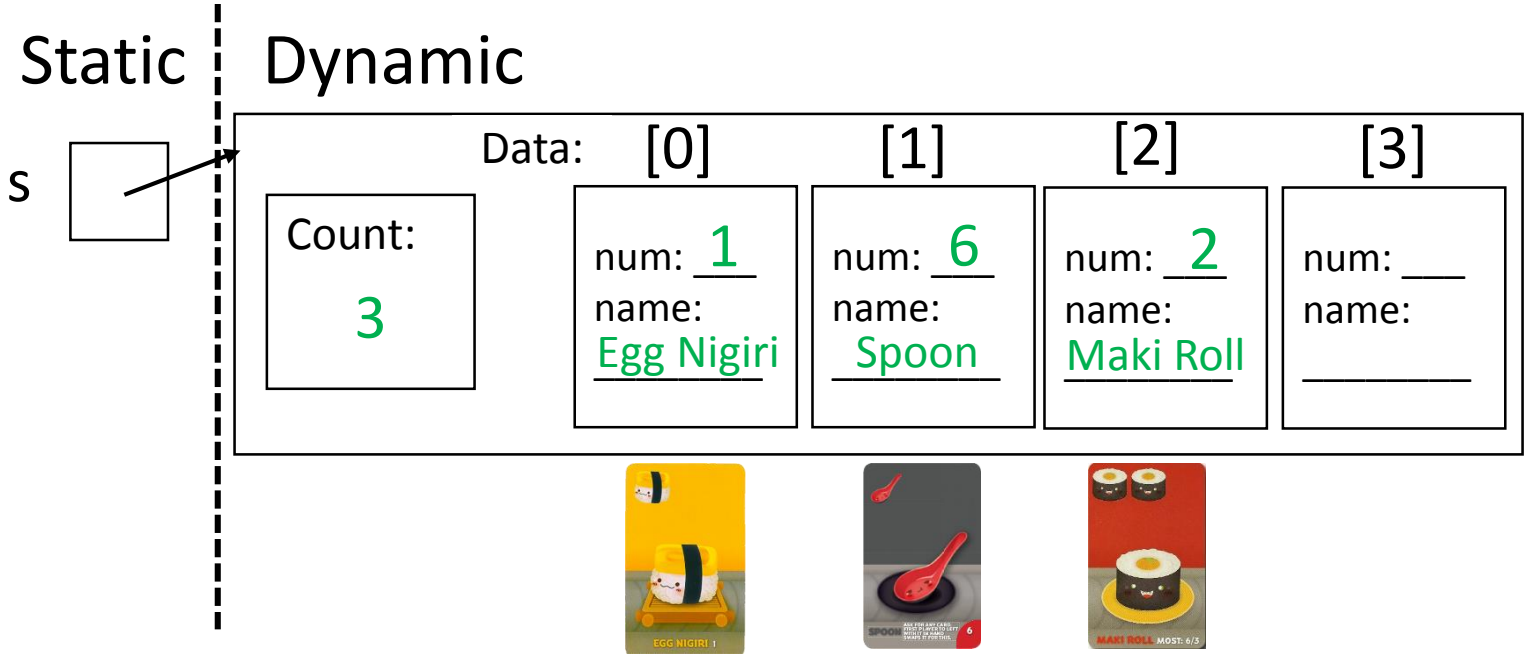

```
public class Stack {

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

public Stack () {
    count = 0;
}

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

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

Create an memory diagram for this code.

```
SushiStack s = new SushiStack ();
s.push (new Sushi (1,"Egg Nigiri"));
s.push (new Sushi (6,"Spoon"));
s.push (new Sushi ());
s.pop();
s.push (new Sushi (4,"Sashimi"));
s.pop();
```

```
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;
   }
```

Memory Diagram



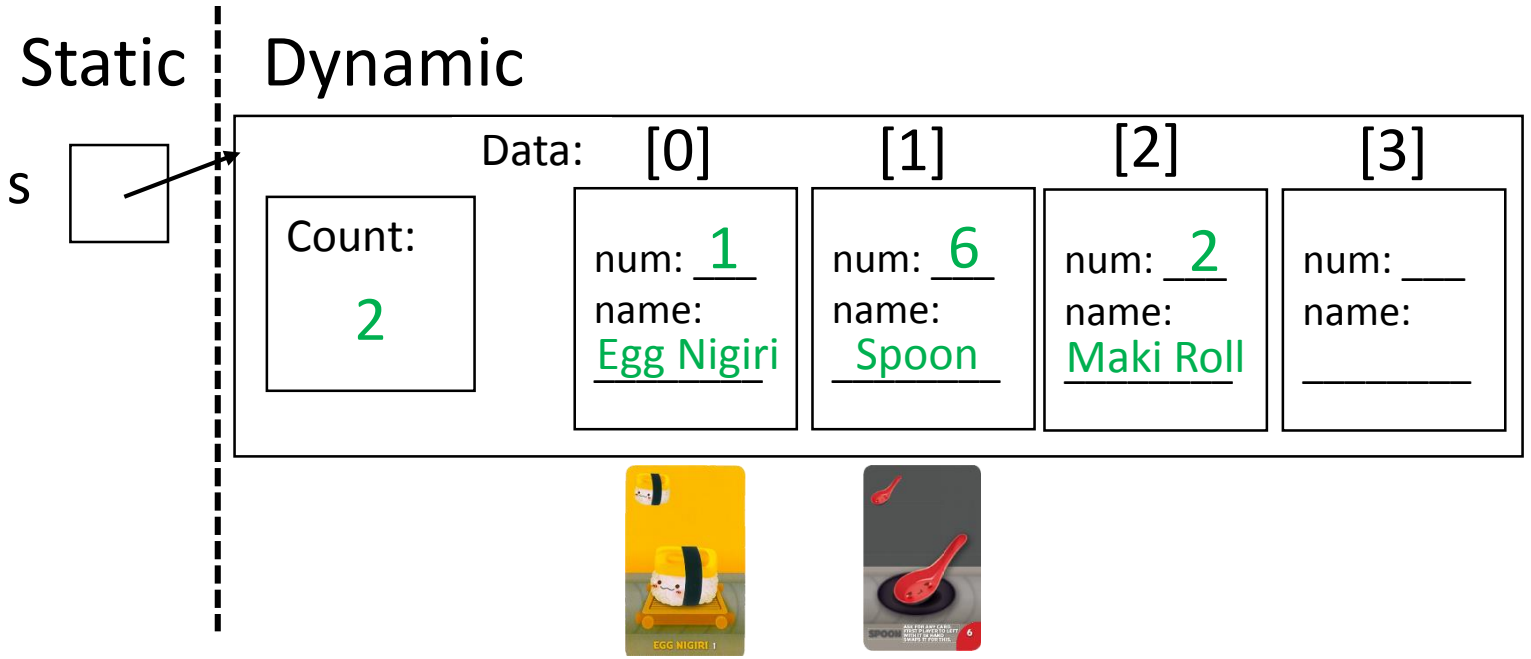

```
public class Stack {

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

public Stack () {
    count = 0;
}

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

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

Create an memory diagram for this code.

```
SushiStack s = new SushiStack ();
s.push (new Sushi (1,"Egg Nigiri"));
s.push (new Sushi (6,"Spoon"));
s.push (new Sushi ());
s.pop();
s.push (new Sushi (4,"Sashimi"));
s.pop();
```

```
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;
   }
```

Memory Diagram



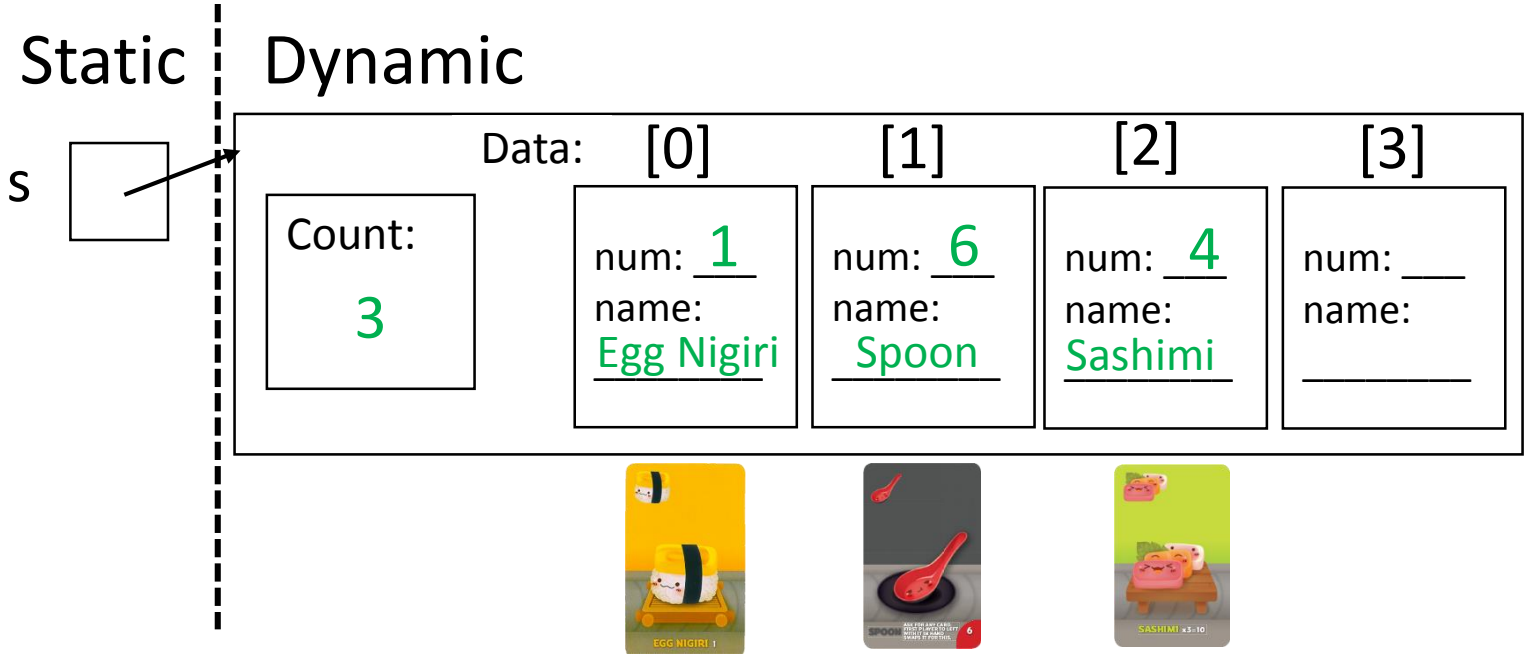

```
public class Stack {

private int count;

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

public Stack () {
    count = 0;
}

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

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

Create an memory diagram for this code.

```
SushiStack s = new SushiStack ();
s.push (new Sushi (1,"Egg Nigiri"));
s.push (new Sushi (6,"Spoon"));
s.push (new Sushi ());
s.pop();
s.push (new Sushi (4,"Sashimi"));
s.pop();
```

```
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;
   }
```

Memory Diagram



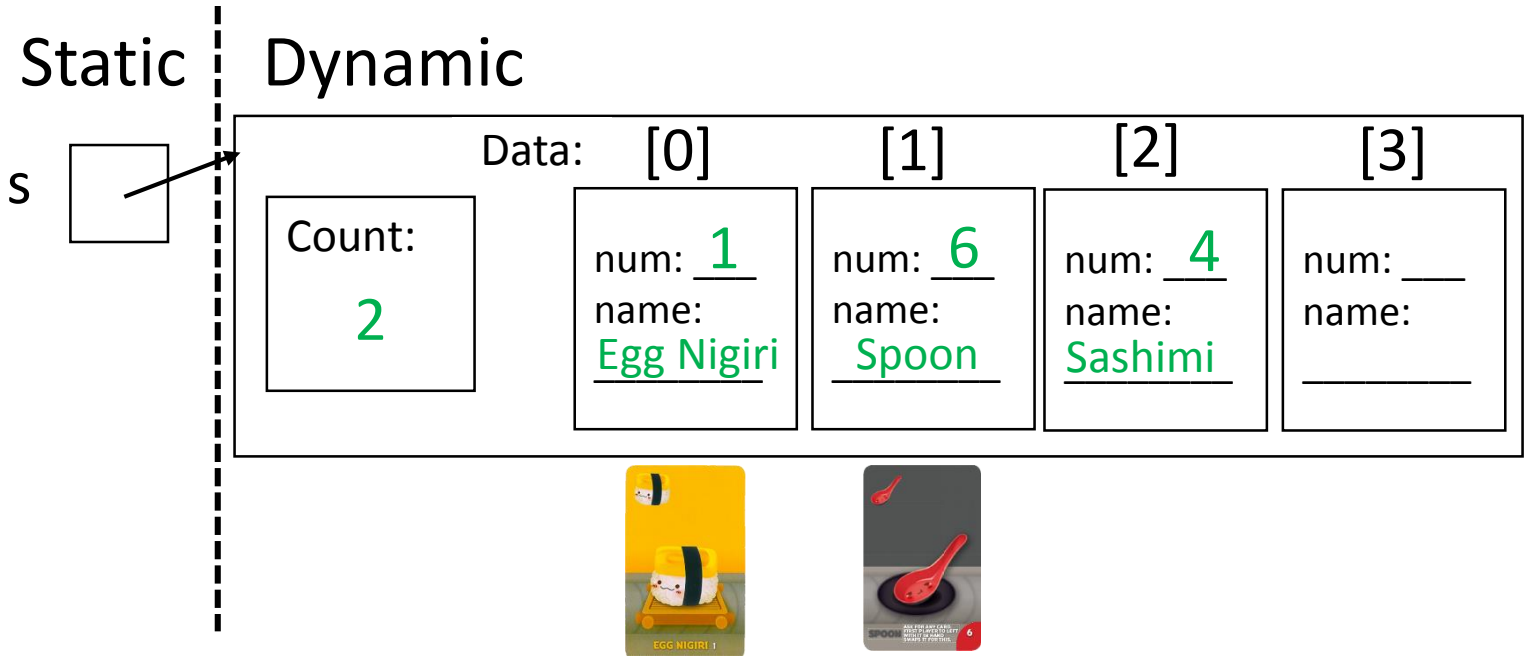

```
public class Stack {

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

public Stack () {
    count = 0;
}

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

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

Create an memory diagram for this code.

```
SushiStack s = new SushiStack ();
s.push (new Sushi (1,"Egg Nigiri"));
s.push (new Sushi (6,"Spoon"));
s.push (new Sushi ());
s.pop();
s.push (new Sushi (4,"Sashimi"));
s.pop();
```

```
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;
   }
```

Memory Diagram




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

Pictures/ Rough Files