CS 10:
Problem solving via Object Oriented Programming

Animated Images

Agenda

1. Multiple blobs: lists
2. Images
3. Animated images

Java provides an ArrayList that can hold a collection of objects

ArrayList

- Stores list of objects in order
- Variable length – don't specify size; can grow (unlike C array, but like Python list)
- Random access – get item by index (starting at 0)
- Must be imported from java.util (Eclipse can help!)
- Provides methods to add or remove elements
- Specify what type of object it holds in angle brackets <> (e.g., ArrayList<Blob> or ArrayList<String>)
- ArrayList called a *generic* container because it can hold any type of object
- All objects must be of same type (unlike Python)

ArrayList methods provide a consistent means of interaction

ArrayList methods

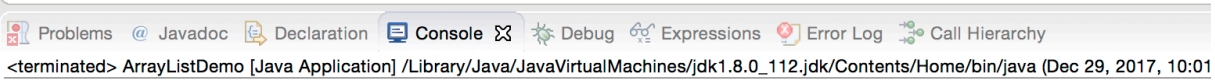
- *add (E elmt)* – appends element *elmt* to end of list
- *add (int index, E elmt)* – inserts element *elmt* at position *index*
- *get (int index)* – returns the element at position *index*
- *remove (int index)* – removes (and returns) the element at position *index*
- *set(int index, E elmt)* – sets item at position *index* to *elmt*
- *size()* – returns the number of elements in the ArrayList
- Others on Oracle website

ArrayListDemo.java: ArrayLists can hold multiple objects, provide useful methods

```
1  import java.util.ArrayList;
2
3  public class ArrayListDemo {
4
5      public static void main(String[] args) {
6          ArrayList<Integer> a = new ArrayList<Integer>();
7          a.add(1);
8          a.add(2);
9          a.add(1,3);
10         System.out.println(a);
11         Integer b = a.get(1);
12         System.out.println(b);
13         a.remove(1);
14         System.out.println(a);
15         a.set(1, 4);
16         System.out.println(a);
17         System.out.println(a.size());
18     }
19 }
20
```

Must import Arraylist

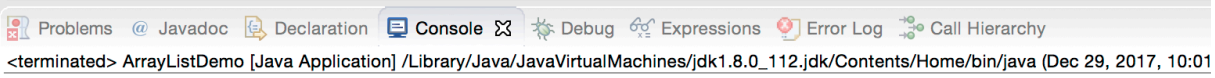
- Provide type of objects ArrayList will hold in <> brackets (can't be primitive)
- Integer is the object version of int
- ArrayLists can hold only ***one*** type of object (unlike Python)
- ArrayLists called generic containers because can hold any type of object (Integers, Doubles, Strings, Blobs)
- Don't need to specify length of ArrayList, it can grow (unlike C array)



ArrayListDemo.java: ArrayLists can hold multiple objects, provide useful methods

```
1  import java.util.ArrayList;
2
3  public class ArrayListDemo {
4
5      public static void main(String[] args) {
6          ArrayList<Integer> a = new ArrayList<Integer>();
7          a.add(1);
8          a.add(2);
9          a.add(1,3);
10         System.out.println(a);
11         Integer b = a.get(1);
12         System.out.println(b);
13         a.remove(1);
14         System.out.println(a);
15         a.set(1, 4);
16         System.out.println(a);
17         System.out.println(a.size());
18     }
19 }
20
```

- ***add(E elmt)* appends item to end of ArrayList**
  - ***E* = type (Integer here)**
  - ***elmt* = object (element) to add to ArrayList**



ArrayListDemo.java: ArrayLists can hold multiple objects, provide useful methods

```
1  import java.util.ArrayList;
2
3  public class ArrayListDemo {
4
5      public static void main(String[] args) {
6          ArrayList<Integer> a = new ArrayList<Integer>();
7          a.add(1);
8          a.add(2);
9          a.add(1,3);
10         System.out.println(a);
11         Integer b = a.get(1);
12         System.out.println(b);
13         a.remove(1);
14         System.out.println(a);
15         a.set(1, 4);
16         System.out.println(a);
17         System.out.println(a.size());
18     }
19 }
20
```

- ***add(E elmt)* appends item to end of ArrayList**
  - ***E* = type (Integer here)**
  - ***elmt* = object (element) to add to ArrayList**

Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> ArrayListDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 29, 2017, 10:01

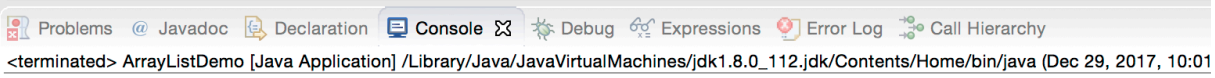
ArrayList a

1

ArrayListDemo.java: ArrayLists can hold multiple objects, provide useful methods

```
1  import java.util.ArrayList;
2
3  public class ArrayListDemo {
4
5      public static void main(String[] args) {
6          ArrayList<Integer> a = new ArrayList<Integer>();
7          a.add(1);
8          a.add(2);
9          a.add(1,3);
10         System.out.println(a);
11         Integer b = a.get(1);
12         System.out.println(b);
13         a.remove(1);
14         System.out.println(a);
15         a.set(1, 4);
16         System.out.println(a);
17         System.out.println(a.size());
18     }
19 }
20
```

- ***add(E elmt)* appends item to end of ArrayList**
  - ***E* = type (Integer here)**
  - ***elmt* = object (element) to add to ArrayList**



ArrayList a

ArrayListDemo.java: ArrayLists can hold multiple objects, provide useful methods

```
1  import java.util.ArrayList;
2
3  public class ArrayListDemo {
4
5      public static void main(String[] args) {
6          ArrayList<Integer> a = new ArrayList<Integer>();
7          a.add(1);
8          a.add(2);
9          a.add(1,3);
10         System.out.println(a);
11         Integer b = a.get(1);
12         System.out.println(b);
13         a.remove(1);
14         System.out.println(a);
15         a.set(1, 4);
16         System.out.println(a);
17         System.out.println(a.size());
18     }
19 }
20
```

- ***add(int i, E elmt)*** **adds item at index *i***
- **ArrayLists are zero indexed (start at index 0, unlike Matlab)**
- **Items slide right to make room**

Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> ArrayListDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 29, 2017, 10:01

ArrayList a

ArrayListDemo.java: ArrayLists can hold multiple objects, provide useful methods

```
1  import java.util.ArrayList;
2
3  public class ArrayListDemo {
4
5      public static void main(String[] args) {
6          ArrayList<Integer> a = new ArrayList<Integer>();
7          a.add(1);
8          a.add(2);
9          a.add(1,3);
10         System.out.println(a);
11         Integer b = a.get(1);
12         System.out.println(b);
13         a.remove(1);
14         System.out.println(a);
15         a.set(1, 4);
16         System.out.println(a);
17         System.out.println(a.size());
18     }
19 }
20
```

- ***add(int i, E elmt)*** **adds item at index *i***
- **ArrayLists are zero indexed (start at index 0, unlike Matlab)**
- **Items slide right to make room**

Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> ArrayListDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 29, 2017, 10:01

ArrayList a

ArrayListDemo.java: ArrayLists can hold multiple objects, provide useful methods

```
1  import java.util.ArrayList;
2
3  public class ArrayListDemo {
4
5      public static void main(String[] args) {
6          ArrayList<Integer> a = new ArrayList<Integer>();
7          a.add(1);
8          a.add(2);
9          a.add(1,3);
10         System.out.println(a);
11         Integer b = a.get(1);
12         System.out.println(b);
13         a.remove(1);
14         System.out.println(a);
15         a.set(1, 4);
16         System.out.println(a);
17         System.out.println(a.size());
18     }
19 }
20
```

- ***add(int i, E elmt)*** **adds item at index *i***
- **ArrayLists are zero indexed (start at index 0, unlike Matlab)**
- **Items slide right to make room**

Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> ArrayListDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 29, 2017, 10:01

```
[1, 3, 2]
```

ArrayList a

1	3	2

ArrayListDemo.java: ArrayLists can hold multiple objects, provide useful methods

```
1  import java.util.ArrayList;
2
3  public class ArrayListDemo {
4
5      public static void main(String[] args) {
6          ArrayList<Integer> a = new ArrayList<Integer>();
7          a.add(1);
8          a.add(2);
9          a.add(1,3);
10         System.out.println(a);
11         Integer b = a.get(1);
12         System.out.println(b);
13         a.remove(1);
14         System.out.println(a);
15         a.set(1, 4);
16         System.out.println(a);
17         System.out.println(a.size());
18     }
19 }
20
```

- **ArrayLists provide random access (can get item from anywhere)**
- ***get(int i)* returns item at index *i***
- **Remember zero-based indexing!**

Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> ArrayListDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 29, 2017, 10:01

```
[1, 3, 2]
```

ArrayList a

ArrayListDemo.java: ArrayLists can hold multiple objects, provide useful methods

```
1  import java.util.ArrayList;
2
3  public class ArrayListDemo {
4
5      public static void main(String[] args) {
6          ArrayList<Integer> a = new ArrayList<Integer>();
7          a.add(1);
8          a.add(2);
9          a.add(1,3);
10         System.out.println(a);
11         Integer b = a.get(1);
12         System.out.println(b);
13         a.remove(1);
14         System.out.println(a);
15         a.set(1, 4);
16         System.out.println(a);
17         System.out.println(a.size());
18     }
19 }
20
```

- **ArrayLists provide random access (can get item from anywhere)**
- ***get(int i)* returns item at index *i***
- **Remember zero-based indexing!**

Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> ArrayListDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 29, 2017, 10:01

```
[1, 3, 2]
3
```

ArrayList a

1	3	2

ArrayListDemo.java: ArrayLists can hold multiple objects, provide useful methods

```
1  import java.util.ArrayList;
2
3  public class ArrayListDemo {
4
5      public static void main(String[] args) {
6          ArrayList<Integer> a = new ArrayList<Integer>();
7          a.add(1);
8          a.add(2);
9          a.add(1,3);
10         System.out.println(a);
11         Integer b = a.get(1);
12         System.out.println(b);
13         a.remove(1);
14         System.out.println(a);
15         a.set(1, 4);
16         System.out.println(a);
17         System.out.println(a.size());
18     }
19 }
20
```

- **Can remove item from anywhere in ArrayList**
- ***remove(int i)* removes item at index *i* and "pushes" remaining items left**

Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> ArrayListDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 29, 2017, 10:01

```
[1, 3, 2]
3
```

ArrayList a

1	3	2

ArrayListDemo.java: ArrayLists can hold multiple objects, provide useful methods

```
1  import java.util.ArrayList;
2
3  public class ArrayListDemo {
4
5      public static void main(String[] args) {
6          ArrayList<Integer> a = new ArrayList<Integer>();
7          a.add(1);
8          a.add(2);
9          a.add(1,3);
10         System.out.println(a);
11         Integer b = a.get(1);
12         System.out.println(b);
13         a.remove(1);
14         System.out.println(a);
15         a.set(1, 4);
16         System.out.println(a);
17         System.out.println(a.size());
18     }
19 }
20
```

- **Can remove item from anywhere in ArrayList**
- ***remove(int i)* removes item at index *i* and "pushes" remaining items left**

Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> ArrayListDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 29, 2017, 10:01

```
[1, 3, 2]
3
[1, 2]
```

ArrayList a

1	3	2

ArrayListDemo.java: ArrayLists can hold multiple objects, provide useful methods

```
1  import java.util.ArrayList;
2
3  public class ArrayListDemo {
4
5      public static void main(String[] args) {
6          ArrayList<Integer> a = new ArrayList<Integer>();
7          a.add(1);
8          a.add(2);
9          a.add(1,3);
10         System.out.println(a);
11         Integer b = a.get(1);
12         System.out.println(b);
13         a.remove(1);
14         System.out.println(a);
15         a.set(1, 4);
16         System.out.println(a);
17         System.out.println(a.size());
18     }
19 }
20
```

- ***set(int i, E elmt)*** **sets the item at index *i* to *elmt***

Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> ArrayListDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 29, 2017, 10:01

```
[1, 3, 2]
3
[1, 2]
```

ArrayList a

ArrayListDemo.java: ArrayLists can hold multiple objects, provide useful methods

```
1  import java.util.ArrayList;
2
3  public class ArrayListDemo {
4
5      public static void main(String[] args) {
6          ArrayList<Integer> a = new ArrayList<Integer>();
7          a.add(1);
8          a.add(2);
9          a.add(1,3);
10         System.out.println(a);
11         Integer b = a.get(1);
12         System.out.println(b);
13         a.remove(1);
14         System.out.println(a);
15         a.set(1, 4);
16         System.out.println(a);
17         System.out.println(a.size());
18     }
19 }
20
```

- ***set(int i, E elmt)*** **sets the item at index *i* to *elmt***

Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> ArrayListDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 29, 2017, 10:01

```
[1, 3, 2]
3
[1, 2]
[1, 4]
```

ArrayList a

ArrayListDemo.java: ArrayLists can hold multiple objects, provide useful methods

```
1  import java.util.ArrayList;
2
3  public class ArrayListDemo {
4
5      public static void main(String[] args) {
6          ArrayList<Integer> a = new ArrayList<Integer>();
7          a.add(1);
8          a.add(2);
9          a.add(1,3);
10         System.out.println(a);
11         Integer b = a.get(1);
12         System.out.println(b);
13         a.remove(1);
14         System.out.println(a);
15         a.set(1, 4);
16         System.out.println(a);
17         System.out.println(a.size());
18     }
19 }
20
```

- ***size()* returns the number of items stored in the ArrayList**

Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> ArrayListDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 29, 2017, 10:01

```
[1, 3, 2]
3
[1, 2]
[1, 4]
```

ArrayList a

1	4

ArrayListDemo.java: ArrayLists can hold multiple objects, provide useful methods

```
1  import java.util.ArrayList;
2
3  public class ArrayListDemo {
4
5      public static void main(String[] args) {
6          ArrayList<Integer> a = new ArrayList<Integer>();
7          a.add(1);
8          a.add(2);
9          a.add(1,3);
10         System.out.println(a);
11         Integer b = a.get(1);
12         System.out.println(b);
13         a.remove(1);
14         System.out.println(a);
15         a.set(1, 4);
16         System.out.println(a);
17         System.out.println(a.size());
18     }
19 }
20
```

- ***size()* returns the number of items stored in the ArrayList**

Problems | Javadoc | Declaration | Console | Debug | Expressions | Error Log | Call Hierarchy

<terminated> ArrayListDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 29, 2017, 10:01

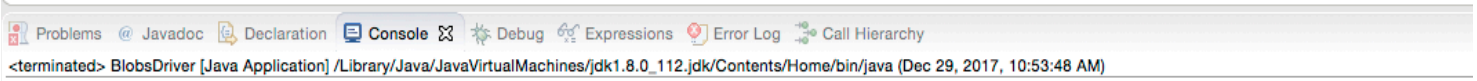
```
[1, 3, 2]
3
[1, 2]
[1, 4]
2
```

ArrayList a

BlobsDriver.java: ArrayList holds multiple Blobs, each of different subclass

```
1  import java.util.ArrayList;
2  /**
3   * Demonstrate ArrayList holding multiple Blobs
4   * @author Chris Bailey-Kellogg
5   * @author Tim Pierson, Dartmouth CS 10, Winter 2018
6   *
7   */
8  public class BlobsDriver {
9
10     public static void main(String[] args) {
11         ArrayList<Blob> blobs = new ArrayList<Blob>();
12         blobs.add(new Wanderer(10,10));
13         blobs.add(new Bouncer(20,30,800,600));
14         blobs.get(0).step(); // => the wanderer
15         blobs.get(1).step(); // => the bouncer
16         System.out.println(blobs.size()); // => 2
17         System.out.println(blobs.get(0).getX());
18     }
19 }
20
```

- **ArrayList to hold multiple Blob objects**



BlobsDriver.java: ArrayList holds multiple Blobs, each of different subclass

```
1  import java.util.ArrayList;
2  /**
3   * Demonstrate ArrayList holding multiple Blobs
4   * @author Chris Bailey-Kellogg
5   * @author Tim Pierson, Dartmouth CS 10, Winter 2018 -- added print statements
6   *
7   */
8  public class BlobsDriver {
9
10     public static void main(String[] args) {
11         ArrayList<Blob> blobs = new ArrayList<Blob>();
12         blobs.add(new Wanderer(10,10));
13         blobs.add(new Bouncer(20,30,800,600));
14         blobs.get(0).step(); // => the wanderer
15         blobs.get(1).step(); // => the bouncer
16         System.out.println(blobs.size()); // => 2
17         System.out.println(blobs.get(0).getX());
18     }
19 }
20
```

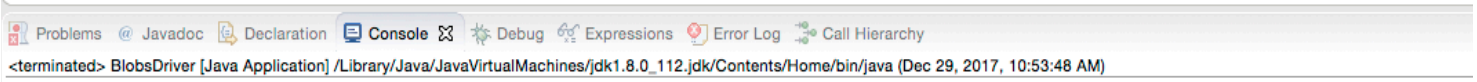
- **ArrayList to hold multiple Blob objects**
- **Because each subclass (e.g., Wanderer, Bouncer) "is a" Blob, Java allows adding objects that are subclasses from Blob base class**

```
Problems  Javadoc  Declaration  Console  Debug  Expressions  Error Log  Call Hierarchy
<terminated> BlobsDriver [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 29, 2017, 10:53:48 AM)
```

BlobsDriver.java: ArrayList holds multiple Blobs, each of different subclass

```
1 import java.util.ArrayList;
2 /**
3  * Demonstrate ArrayList holding multiple Blobs
4  * @author Chris Bailey-Kellogg
5  * @author Tim Pierson, Dartmouth CS 10, Winter 2018 -- added print statements
6  *
7  */
8 public class BlobsDriver {
9
10     public static void main(String[] args) {
11         ArrayList<Blob> blobs = new ArrayList<Blob>();
12         blobs.add(new Wanderer(10,10));
13         blobs.add(new Bouncer(20,30,800,600));
14         blobs.get(0).step(); // => the wanderer
15         blobs.get(1).step(); // => the bouncer
16         System.out.println(blobs.size()); // => 2
17         System.out.println(blobs.get(0).getX());
18     }
19 }
20
```

- ArrayList to hold multiple Blob objects
- Because each subclass (e.g., Wanderer, Bouncer) "is a" Blob, Java allows adding objects that are subclasses from Blob base class
- Use *get(i)* to access Blobs stored in ArrayList at index *i*



BlobsDriver.java: ArrayList holds multiple Blobs, each of different subclass

```
1  import java.util.ArrayList;
2  /**
3   * Demonstrate ArrayList holding multiple Blobs
4   * @author Chris Bailey-Kellogg
5   * @author Tim Pierson, Dartmouth CS 10, Winter 2018 -- added print statements
6   *
7   */
8  public class BlobsDriver {
9
10     public static void main(String[] args) {
11         ArrayList<Blob> blobs = new ArrayList<Blob>();
12         blobs.add(new Wanderer(10,10));
13         blobs.add(new Bouncer(20,30,800,600));
14         blobs.get(0).step(); // => the wanderer
15         blobs.get(1).step(); // => the bouncer
16         System.out.println(blobs.size()); // => 2
17         System.out.println(blobs.get(0).getX());
18     }
19 }
20
```

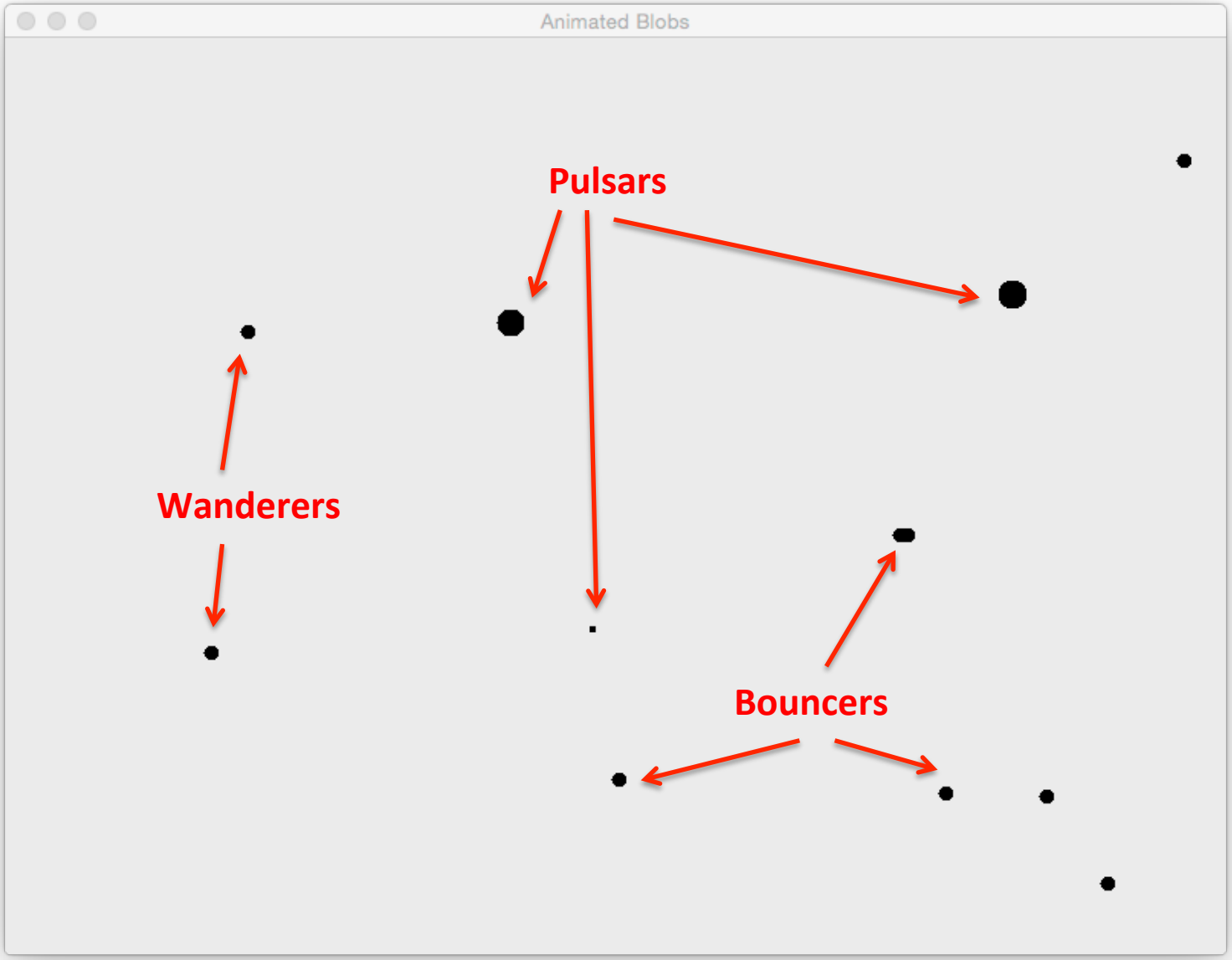
- **ArrayList to hold multiple Blob objects**
- **Because each subclass (e.g., Wanderer, Bouncer) "is a" Blob, Java allows adding objects that are subclasses from Blob base class**
- **Use *get(i)* to access Blobs stored in ArrayList at index *i***
- **Can call object methods after getting (e.g., call *getX()* after getting Blob at index 0)**

Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> BlobsDriver [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 29, 2017, 10:53:48 AM)

```
2
10.273250985689945
```

BlobsGUI.java uses an ArrayList to store multiple Blobs



BlobsGUI.java uses an ArrayList to store multiple Blobs

```
10 public class BlobsGUI extends DrawingGUI {
11     private static final int width=800, height=600;  // size of the world
12
13     private ArrayList<Blob> blobs;                   // list of all the blobs to handle
14     private char blobType = '0';                     // what type of blob to create
15     private int delay = 100;                         // for the timer
16
17     public BlobsGUI() {
18         super("Animated Blobs", width, height);
19
20         // Initialize empty list of blobs. What happens if we forget to do this?
21         // (You will run into that situation in the future, I guarantee.)
22         blobs = new ArrayList<Blob>();
23
24         // Timer drives the animation.
25         startTimer();
26     }
27
28     /**
29      * DrawingGUI method, here either detecting which blob was clicked,
30      * or creating a new blob.
31      */
32     @Override
33     public void handleMousePress(int x, int y) {
34         // Check if hit a blob
35         for (Blob blob : blobs) {
36             if (blob.contains(x, y)) {
37                 System.out.println("back off!");
38                 return;
39             }
40         }
41
42         // Create a new blob
43         if (blobType=='0') {
44             blobs.add(new Blob(x,y));
45         }
46         else if (blobType=='b') {
47             blobs.add(new Bouncer(x,y,width,height));
48         }
49         else if (blobType=='p') {
50             blobs.add(new Pulsar(x,y));
```

- **ArrayList to hold multiple Blob objects (previously only one Blob)**
- ***blobType* keeps track of what kind of blob to add next (changed on keypress)**

BlobsGUI.java uses an ArrayList to store multiple Blobs

```
10 public class BlobsGUI extends DrawingGUI {
11     private static final int width=800, height=600;  // size of the world
12
13     private ArrayList<Blob> blobs;                   // list of all the blobs to handle
14     private char blobType = '0';                     // what type of blob to create
15     private int delay = 100;                         // for the timer
16
17     public BlobsGUI() {
18         super("Animated Blobs", width, height);
19
20         // Initialize empty list of blobs. What happens if we forget to do this?
21         // (You will run into that situation in the future, I guarantee.)
22         blobs = new ArrayList<Blob>();
23
24         // Timer drives the animation.
25         startTimer();
26     }
27
28     /**
29      * DrawingGUI method, here either detecting which blob was clicked,
30      * or creating a new blob.
31      */
32     @Override
33     public void handleMousePress(int x, int y) {
34         // Check if hit a blob
35         for (Blob blob : blobs) {
36             if (blob.contains(x, y)) {
37                 System.out.println("back off!");
38                 return;
39             }
40         }
41
42         // Create a new blob
43         if (blobType=='0') {
44             blobs.add(new Blob(x,y));
45         }
46         else if (blobType=='b') {
47             blobs.add(new Bouncer(x,y,width,height));
48         }
49         else if (blobType=='p') {
50             blobs.add(new Pulsar(x,y));
```

- **On mouse press, check to see if mouse is inside *any* Blob (return if true)**
- **This is a for-each loop**
- **Read as "for each blob in blobs"**
- **Loops through each blob object in ArrayList blobs, one at a time**
- **Same as Python (strange for C folks)**

BlobsGUI.java uses an ArrayList to store multiple Blobs

```
10 public class BlobsGUI extends DrawingGUI {
11     private static final int width=800, height=600;  // size of the world
12
13     private ArrayList<Blob> blobs;                   // list of all the blobs to handle
14     private char blobType = '0';                     // what type of blob to create
15     private int delay = 100;                         // for the timer
16
17     public BlobsGUI() {
18         super("Animated Blobs", width, height);
19
20         // Initialize empty list of blobs. What happens if we forget to do this?
21         // (You will run into that situation in the future, I guarantee.)
22         blobs = new ArrayList<Blob>();
23
24         // Timer drives the animation.
25         startTimer();
26     }
27
28     /**
29      * DrawingGUI method, here either detecting which blob was clicked,
30      * or creating a new blob.
31      */
32     @Override
33     public void handleMousePress(int x, int y) {
34         // Check if hit a blob
35         for (Blob blob : blobs) {
36             if (blob.contains(x, y)) {
37                 System.out.println("back off!");
38                 return;
39             }
40         }
41
42         // Create a new blob
43         if (blobType=='0') {
44             blobs.add(new Blob(x,y));
45         }
46         else if (blobType=='b') {
47             blobs.add(new Bouncer(x,y,width,height));
48         }
49         else if (blobType=='p') {
50             blobs.add(new Pulsar(x,y));
```

- On mouse press, check to see if mouse is inside any Blob (return if true)
- This is a for-each loop
- Read as “for each blob in blobs”
- Loops through each blob object in ArrayList blobs, one at a time
- Same as Python (strange for C folks)
- Add new blob to ArrayList each time mouse clicked if mouse not inside any Blob

BlobsGUI.java uses an ArrayList to store multiple Blobs

```
86
87      /**
88       * DrawingGUI method, here just drawing all the blobs
89       */
90      @Override
91      public void draw(Graphics g) {
92          // Ask all the blobs to draw themselves.
93          for (Blob blob : blobs) {
94              blob.draw(g);
95          }
96      }
97
98      /**
99       * DrawingGUI method, here having all the blobs take a step
100      */
101     @Override
102     public void handleTimer() {
103         // Ask all the blobs to move themselves.
104         for (Blob blob : blobs) {
105             blob.step();
106         }
107         // Now update the GUI.
108         repaint();
109     }
110
```

***draw()* and *handleTimer()* each use the for-each loop to draw and step each of the Blobs in the ArrayList**

Agenda

1. Multiple blobs: lists
2. Images
3. Animated images

SmileGUI.java: DrawingGUI has method to load and store images in a BufferedImage

```
1  import java.awt.Graphics;
5
6  /**
7   * Simple illustration of GUI to display an image
8   *
9   * @author Chris Bailey-Kellogg, Dartmouth CS 10, Spring 2016
10  */
11 public class SmileGUI extends DrawingGUI {
12     private BufferedImage img;
13
14     public SmileGUI(BufferedImage img) {
15         super("Smile!", img.getWidth(), img.getHeight());
16         this.img = img;
17     }
18
19     /**
20      * DrawingGUI method, here showing the image
21      */
22     @Override
23     public void draw(Graphics g) {
24         g.drawImage(img, 0, 0, null);
25     }
26
27     public static void main(String[] args) {
28         SwingUtilities.invokeLater(new Runnable() {
29             public void run() {
30                 // Load image into memory from disk
31                 BufferedImage img = loadImage("pictures/smiley.png");
32
33                 // Create a GUI to display it
34                 new SmileGUI(img);
35             }
36         });
37     }
38 }
39
```

BufferedImage **instance variable to hold image that will be loaded from disk**

Store image in instance variable

Override *draw()* to show image instead of asking each Blob to draw itself
Remember *draw()* is called by *repaint()*

Load image from disk with *loadImage()* method

Call constructor, passing newly loaded image

SmileGUI.java: DrawingGUI has method to load and store images in a BufferedImage

```java
import java.awt.Graphics;

/**
 * Simple illustration of GUI to display an image
 *
 * @author Chris Bailey-Kellogg, Dartmouth CS 10, Spring 2016
 */
public class SmileGUI extends DrawingGUI {
    private BufferedImage img;

    public SmileGUI(BufferedImage img) {
        super("Smile!", img.getWidth(), img.getHeight());
        this.img = img;
    }

    /**
     * DrawingGUI method, here showing the image
     */
    @Override
    public void draw(Graphics g) {
        g.drawImage(img, 0, 0, null);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                // Load image into memory from disk
                BufferedImage img = loadImage("pictures/smiley.png");

                // Create a GUI to display it
                new SmileGUI(img);
            }
        });
    }
}
```

Result

WanderingImage.java: Creates a new Blob subclass that shows image instead of oval

```
1  import java.awt.Graphics;
3
4  /**
5   * A blob that moves randomly and draws itself as an image
6   */
7  public class WanderingImage extends Wanderer {
8      private BufferedImage img;
9
10     public WanderingImage(double x, double y, BufferedImage img) {
11         super(x, y, Math.max(img.getHeight()/2, img.getWidth()/2));
12         this.img = img;
13     }
14
15     @Override
16     public void draw(Graphics g) {
17         g.drawImage(img, (int)(x-r), (int)(y-r), null);
18     }
19 }
20
```

Inherit from Wanderer, no need to duplicate code

Store image in *BufferedImage* instance variable

Call Wander constructor
Set r based on max of height or width

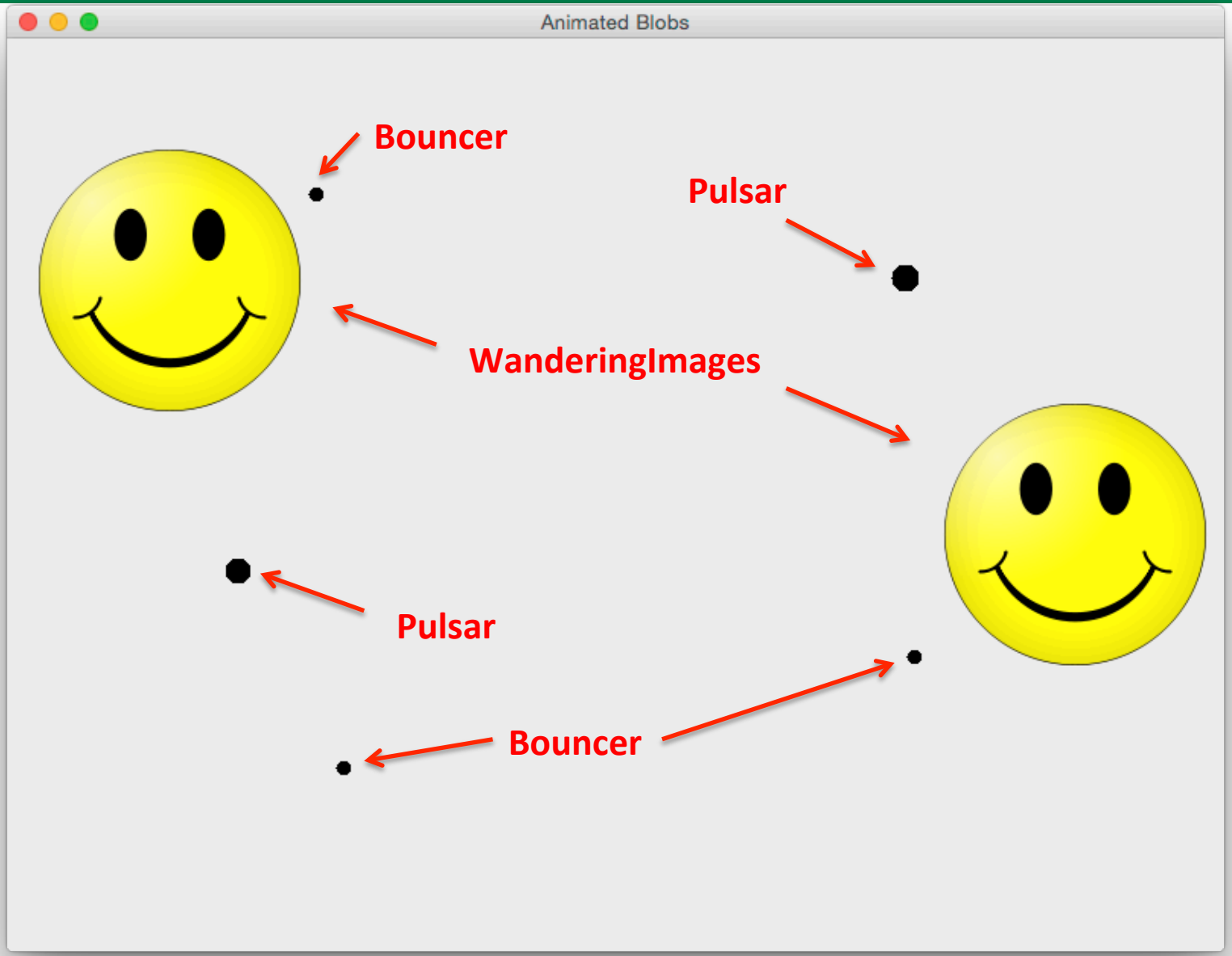
Save image in constructor

Override Wanderer's *draw()* function to show image instead of calling *fillOval()*

Note how little code has to change for the Blob to do something completely different (this is all the code for the WanderingImage class)

BlobGUI2.java makes minor tweak to add ability to use WanderingImage Blobs on graphics

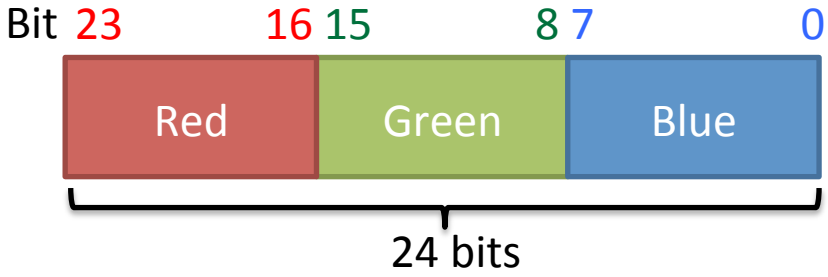
BlobGUI2: Can now add WanderingImage Blobs



Agenda

1. Multiple blobs: lists
2. Images
3. Animated images

Java can represent colors as a 24-bit integer



Java can use 24-bit integer to represent red, green, and blue color component intensity

Each color component has 8 bits, so intensity range for each component is 0-255:

0 = no color

255 = max color

Java provides a convenient Color class to store color values

WanderingPixel.java: Makes colored Blobs using Java's Color class

```
1  import java.awt.*;
2
3  /**
4   * A blob that moves randomly and has a color
5   */
6  public class WanderingPixel extends Wanderer {
7      private Color color;
8
9      public WanderingPixel(double x, double y, double r, Color c) {
10         super(x, y, r);
11         this.color = c;
12     }
13
14     @Override
15     public void draw(Graphics g) {
16         g.setColor(color);
17         g.fillOval((int)(x-r), (int)(y-r), (int)(2*r), (int)(2*r));
18     }
19 }
20
```

Inherit from Wanderer, no need to duplicate code

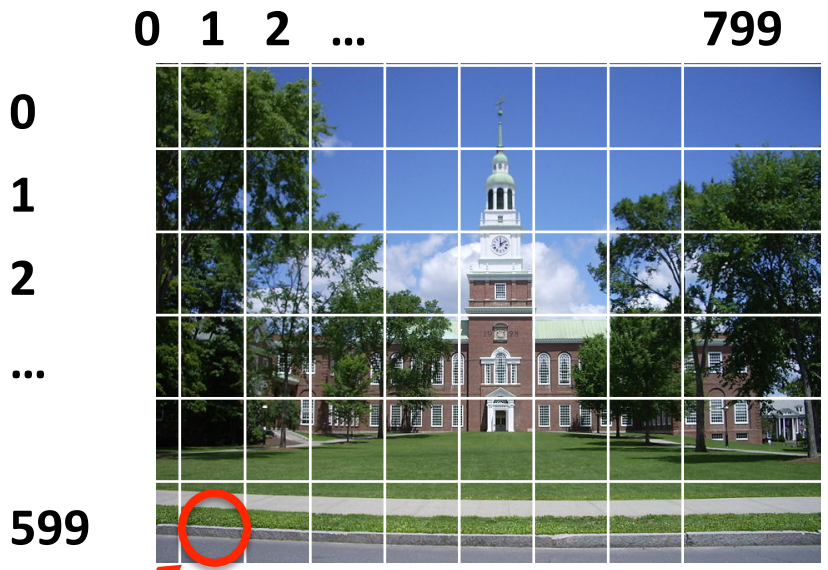
Store Blob color in instance variable

Set color in *draw()*

Images are made up of pixels, each with a (x,y) location and a color

800 x 600 image

NOTE Y axis counts downward!



We can get and set image pixel colors (`img` is BufferedImage)

`img.getRGB(x,y)` – returns Color of pixel at `x,y`

`img.setRGB(x,y,color)` – sets Color of pixel at `x,y`

AnimatedImage.java: Now we can have some fun with images

Pseudo code

- Load image
- Create 20,000 WanderingPixels (colored blobs)
  - Randomly pick x,y location
  - Use `getRGB(x,y)` to pick up image color at x,y
  - Set blob to that color
- At each timer tick, randomly choose 1,000 blob to step()
- Image jitters
- AnimatedImage.java has actual code

AnimatedImage.java: Now we can have some fun with images

```
14 public class AnimatedImage extends DrawingGUI {
15     private static final int radius = 5;                // setup: blob size
16     private static final int numBlobs = 20000;          // setup: how many blobs
17     private static final int numToMove = 1000;          // setup: how many blobs to animate each frame
18
19     private ArrayList<Blob> blobs;                      // the blobs representing the picture
20
21     public AnimatedImage(BufferedImage img) {
22         // Size of the window is scaled up from the original image by the blob radius
23         super("Animated Picture", img.getWidth()*radius, img.getHeight()*radius);
24
25         // Create a bunch of random blobs.
26         blobs = new ArrayList<Blob>();
27         for (int i=0; i<numBlobs; i++) {
28             int x = (int)(img.getWidth()*Math.random());
29             int y = (int)(img.getHeight()*Math.random());
30             Color color = new Color(img.getRGB(x,y));
31             blobs.add(new WanderingPixel(x*radius, y*radius, (int)(1+Math.random()*radius), color));
32         }
33
34         // Timer drives the animation.
35         startTimer();
36     }
37
38     /**
39      * DrawingGUI method, here just drawing all the blobs
40      */
41     @Override
42     public void draw(Graphics g) {
43         for (Blob blob : blobs) {
44             blob.draw(g);
45         }
46     }
47
48     /**
49      * DrawingGUI method, here moving some of the blobs
50      */
51     @Override
52     public void handleTimer() {
53         for (int b = 0; b < numToMove; b++) {
54             // Pick a random blob and ask it to move.
55             blobs.get((int)(Math.random()*blobs.size())).step();
56         }
57         // Now update the drawing
58         repaint();
59     }
60
61     public static void main(String[] args) {
62         SwingUtilities.invokeLater(new Runnable() {
63             public void run() {
64                 // note: using downsampled version, to reduce computational expense
65                 new AnimatedImage(loadImage("pictures/baker-200-150.jpg"));
66             }
67         });
68     }
```

Extend *DrawingGUI* to get functionality

ArrayList called *blobs* will hold 20,000 Blobs

Set up graphics screen

Create 20,000 Wandering Pixels (colored Blobs)
Get blob color from random x,y location on image
Add new Wandering Pixel to blobs ArrayList

Start timer ticking

On each timer tick, randomly select 1,000 Wandering Pixels, then have those *step()*

***repaint()* screen when done**

Load background image
Pass image to constructor

PaintedImage.java: Fade in image using two BufferedImages

Pseudo code

- Load image into `BufferedImage` called `original`
- Create blank `BufferedImage` called `result` and display `result` on screen
- Create 20,000 Wanders at random x,y locations
- At each timer tick, randomly choose 5,000 Wanders
  - Pick up color from `original` image at Wander's x,y location with `getRBG(x,y)`
  - Copy color to `result` image with `setRGB(x,y,color)`
  - Step Wander to move to nearby x,y location
- Repaint result image
- PaintedImage.java has actual code

PaintedImage.java: Fade in image using two BufferedImages

```
11 public class PaintedImage extends DrawingGUI {
12     private static final int numBlobs = 20000;        // setup: how many blobs
13     private static final int numToMove = 5000;        // setup: how many blobs to animate each frame
14
15     private BufferedImage original;                   // the picture to paint
16     private BufferedImage result;                     // the picture being painted
17     private ArrayList<Blob> blobs;                    // the blobs representing the picture
18
19     public PaintedImage(BufferedImage original) {
20         super("Animated Picture", original.getWidth(), original.getHeight());
21
22         this.original = original;
23         result = new BufferedImage(original.getWidth(), original.getHeight(), BufferedImage.TYPE_INT_ARGB);
24
25         // Create a bunch of random blobs.
26         blobs = new ArrayList<Blob>();
27         for (int i=0; i<numBlobs; i++) {
28             int x = (int)(original.getWidth()*Math.random());
29             int y = (int)(original.getHeight()*Math.random());
30             blobs.add(new Wanderer(x, y, 1));
31         }
32
33         // Timer drives the animation.
34         startTimer();
35     }
36
37     /**
38      * DrawingGUI method, here just drawing all the blobs
39      */
40     @Override
41     public void draw(Graphics g) {
42         g.drawImage(result, 0, 0, null);
43         for (Blob blob : blobs) {
44             blob.draw(g);
45         }
46     }
47
48     /**
49      * DrawingGUI method, here moving some of the blobs
50      */
51     @Override
52     public void handleTimer() {
53         for (int b = 0; b < numToMove; b++) {
54             // Pick a random blob, leave a trail where it is, and ask it to move.
55             Blob blob = blobs.get((int)(Math.random()*blobs.size()));
56             int x = (int)blob.getX(), y = (int)blob.getY();
57             // Careful to stay within the image
58             if (x>=0 && x<width && y>=0 && y<height) {
59                 result.setRGB(x, y, original.getRGB(x, y));
60             }
61             blob.step();
62         }
63         // Now update the drawing
64         repaint();
65     }
66
```

Extend *DrawingGUI* to get functionality

Set up GUI

Save original image

Create new image called *result* (initially blank)

result will image will fade in original image

Create 20,000 Wander Blobs at random x,y locations with r=1

Start timer ticking

Show *result* image (was initially blank, but 5,000 pixels colored in every timer tick)

blobs show up as black dots

On timer tick choose 5,000 random Blobs

Get *color* from *original* image at Blob's *x,y* location

Copy *color* to *result* image at *x,y* location

step() blobs, then repaint() when done (calls *draw()*)

result image gets colored in over time

main() loads image (code not shown on this slide), passes to constructor