

CS 10: Problem solving via Object Oriented Programming

Animated Blobs

Agenda

- 
1. Objects vs. primitives
 2. Inheritance
 3. Quick review of Blobs
 4. Using Inheritance to create different types of Blobs
 5. Graphical User Interface (GUI) programming

Java defines a number of primitive types, each of fixed memory size

Common primitive types

Type	Description	Size	Examples
int	Integer values (no decimal component)	32 bits (4 bytes)	-104,...1,2,3...107,...5032..
double	Double precision floating point (has decimal component)	64 bits (8 bytes)	-123.45, 1.6
boolean	true or false	1 bit	true, false
char	Characters	16 bits (2 bytes for Unicode)	'a','b','...'z'

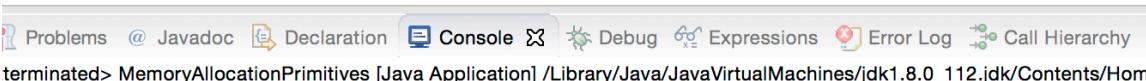
NOTE: Java provides an Object wrapper for each primitive (called autoboxing). Reference them with an initial capital letter (e.g., Integer, Double, Boolean, Character)

Declaring a primitive variable allocates stack space that holds variable's value

Stack

Heap

```
6 public class MemoryAllocationPrimitives {  
7  
8     public static void main(String[] args) {  
9         //declare local variables  
10        int i; double d; boolean b; char c;  
11  
12        //assign values to local variables  
13        i=7; d=1.6; b=true; c='a';  
14  
15        //print new values  
16        System.out.println("Local variables: "+  
17                    "i="+i+" d="+d+" b="+b+" c="+c);  
18    }  
19 }
```



Declaring a primitive variable allocates stack space that holds variable's value

```
6 public class MemoryAllocationPrimitives {  
7  
8     public static void main(String[] args) {  
9         //declare local variables  
10        int i; double d; boolean b; char c;  
11  
12        //assign values to local variables  
13        i=7; d=1.6; b=true; c='a';  
14  
15        //print new values  
16        System.out.println("Local variables: "+  
17                    "i="+i+" d="+d+" b="+b+" c="+c);  
18    }  
19 }
```

Problems @ Javadoc Declaration Console ✎ Debug Expressions Error Log Call Hierarchy
terminated> MemoryAllocationPrimitives [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Hom

Stack Heap

i ?

- While executing line 10, space is allocated on the stack for the primitive local variables
- Java doesn't initialize local variables (like it does instance variables)
- Exception (error) raised if try to use a local variable before value assigned

Declaring a primitive variable allocates stack space that holds variable's value

```
6 public class MemoryAllocationPrimitives {  
7  
8     public static void main(String[] args) {  
9         //declare local variables  
10        int i; double d; boolean b; char c;  
11  
12        //assign values to local variables  
13        i=7; d=1.6; b=true; c='a';  
14  
15        //print new values  
16        System.out.println("Local variables: "+  
17                    "i="+i+" d="+d+" b="+b+" c="+c);  
18    }  
19 }
```

Problems @ Javadoc Declaration Console ✎ Debug Expressions Error Log Call Hierarchy
terminated> MemoryAllocationPrimitives [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Hom

Stack



Heap

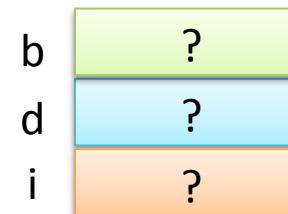
- While executing line 10, space is allocated on the stack for the primitive local variables
- Java doesn't initialize local variables (like it does instance variables)
- Exception (error) raised if try to use a local variable before value assigned
- NOTE: showing primitive types as same size for convenience

Declaring a primitive variable allocates stack space that holds variable's value

```
6 public class MemoryAllocationPrimitives {  
7  
8     public static void main(String[] args) {  
9         //declare local variables  
10        int i; double d; boolean b; char c;  
11  
12        //assign values to local variables  
13        i=7; d=1.6; b=true; c='a';  
14  
15        //print new values  
16        System.out.println("Local variables: "+  
17                    "i="+i+" d="+d+" b="+b+" c="+c);  
18    }  
19 }
```

Problems @ Javadoc Declaration Console ✎ Debug Expressions Error Log Call Hierarchy
terminated> MemoryAllocationPrimitives [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Hom

Stack



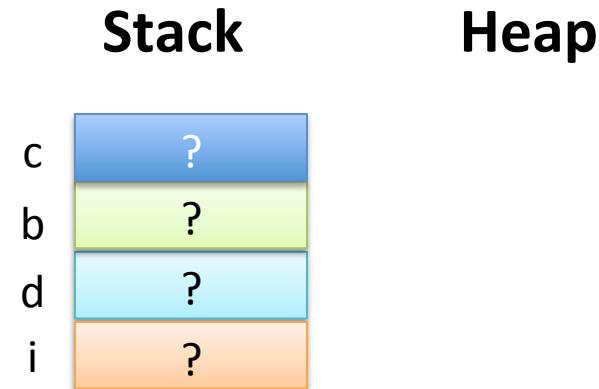
Heap

- While executing line 10, space is allocated on the stack for the primitive local variables
- Java doesn't initialize local variables (like it does instance variables)
- Exception (error) raised if try to use a local variable before value assigned
- NOTE: showing primitive types as same size for convenience

Declaring a primitive variable allocates stack space that holds variable's value

```
6 public class MemoryAllocationPrimitives {  
7  
8     public static void main(String[] args) {  
9         //declare local variables  
10        int i; double d; boolean b; char c;  
11  
12        //assign values to local variables  
13        i=7; d=1.6; b=true; c='a';  
14  
15        //print new values  
16        System.out.println("Local variables: "+  
17                    "i="+i+" d="+d+" b="+b+" c="+c);  
18    }  
19 }
```

Problems @ Javadoc Declaration Console ✎ Debug Expressions Error Log Call Hierarchy
terminated> MemoryAllocationPrimitives [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Hom

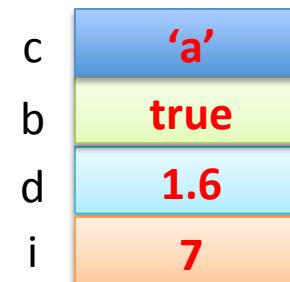


- While executing line 10, space is allocated on the stack for the primitive local variables
- Java doesn't initialize local variables (like it does instance variables)
- Exception (error) raised if try to use a local variable before value assigned
- NOTE: showing primitive types as same size for convenience

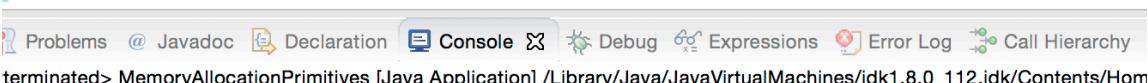
Declaring a primitive variable allocates stack space that holds variable's value

```
6 public class MemoryAllocationPrimitives {  
7  
8     public static void main(String[] args) {  
9         //declare local variables  
10        int i; double d; boolean b; char c;  
11  
12        //assign values to local variables  
13        i=7; d=1.6; b=true; c='a';  
14  
15        //print new values  
16        System.out.println("Local variables: "+  
17                    "i="+i+" d="+d+" b="+b+" c="+c);  
18    }  
19 }
```

Stack Heap



- After executing line 13, values assigned to primitive local variables

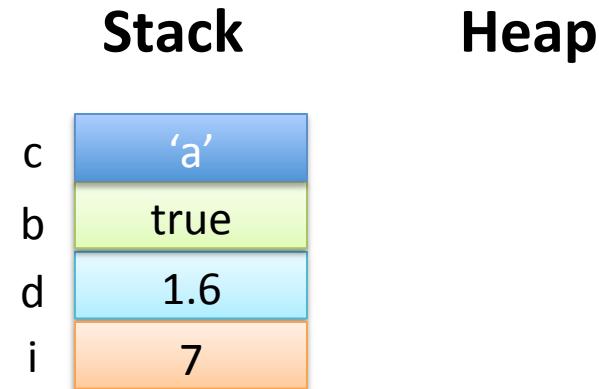


Declaring a primitive variable allocates stack space that holds variable's value

```
6 public class MemoryAllocationPrimitives {  
7  
8     public static void main(String[] args) {  
9         //declare local variables  
10        int i; double d; boolean b; char c;  
11  
12        //assign values to local variables  
13        i=7; d=1.6; b=true; c='a';  
14  
15        //print new values  
16        System.out.println("Local variables: "+  
17                    "i="+i+" d="+d+" b="+b+" c="+c);  
18    }  
19 }
```



```
Problems @ Javadoc Declaration Console ✎ Debug Expressions Error Log Call Hierarchy  
terminated> MemoryAllocationPrimitives [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Hom  
local variables: i=7 d=1.6 b=true c=a
```



- Stack holds the values of the primitive data types
- Printing a primitive type prints its value

Declaring objects makes pointer on the stack, but object itself is elsewhere

```
8 public static void main(String[] args) {  
9     //declare Blob objects  
10    Blob alice = new Blob();  
11    Blob bob; //notice no new keyword  
12    bob = alice; //bob equals alice  
13    Blob charlie = new Blob();  
14    System.out.println("alice.x="+alice.x+  
15                      " bob.x="+bob.x);  
16  
17    //update alice's x  
18    alice.setX(3);  
19    System.out.println("alice.x="+alice.x+  
20                      " bob.x="+bob.x);  
21  
22    //printing objects implicitly calls toString()  
23    System.out.println("alice="+alice+  
24                      " bob="+bob+" charlie="+charlie);  
25 }  
26 }
```

Stack

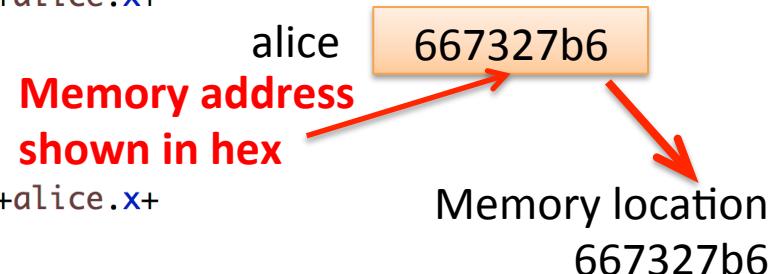
Heap

Declaring objects makes pointer on the stack, but object itself is elsewhere

```
8 public static void main(String[] args) {  
9     //declare Blob objects  
10    Blob alice = new Blob();  
11    Blob bob; //notice no new keyword  
12    bob = alice; //bob equals alice  
13    Blob charlie = new Blob();  
14    System.out.println("alice.x="+alice.x+  
15        " bob.x="+bob.x);  
  
16    //update alice's x  
17    alice.setX(3);  
18    System.out.println("alice.x="+alice.x+  
19        " bob.x="+bob.x);  
  
22    //printing objects implicitly calls toString()  
23    System.out.println("alice="+alice+  
24        " bob="+bob+" charlie="+charlie);  
25 }  
26 }
```

Stack

Heap



r=5

y=0

x=0

- After line 10, stack holds memory address of object (with primitives, stack holds variable's *value*)
- Memory address tells Java where to find the “alice” object in memory
- Object itself allocated elsewhere in memory (in heap, not on stack)
- OS chooses where to allocate

Declaring objects makes pointer on the stack, but object itself is elsewhere

```
8 public static void main(String[] args) {  
9     //declare Blob objects  
10    Blob alice = new Blob();  
11    Blob bob; //notice no new keyword  
12    bob = alice; //bob equals alice  
13    Blob charlie = new Blob();  
14    System.out.println("alice.x="+alice.x+  
15        " bob.x="+bob.x);  
16  
17    //update alice's x  
18    alice.setX(3);  
19    System.out.println("alice.x="+alice.x+  
20        " bob.x="+bob.x);  
21  
22    //printing objects implicitly calls toString()  
23    System.out.println("alice="+alice+  
24        " bob="+bob+" charlie="+charlie);  
25 }  
26 }
```

Stack

bob
alice



Heap

Memory location
667327b6

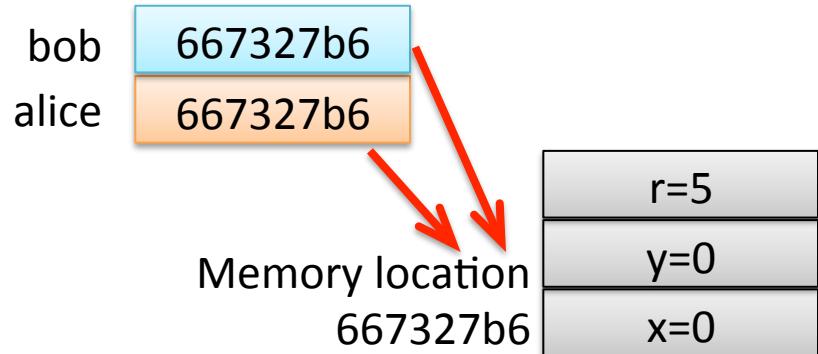
r=5
y=0
x=0

- After line 11, “bob” is allocated on the stack, but is null (points nowhere)
- This is because bob did not use the “new” keyword
- Null pointer exception if try to use bob now

Declaring objects makes pointer on the stack, but object itself is elsewhere

```
8 public static void main(String[] args) {  
9     //declare Blob objects  
10    Blob alice = new Blob();  
11    Blob bob; //notice no new keyword  
12    bob = alice; //bob equals alice  
13    Blob charlie = new Blob();  
14    System.out.println("alice.x="+alice.x+  
15        " bob.x="+bob.x);  
16  
17    //update alice's x  
18    alice.setX(3);  
19    System.out.println("alice.x="+alice.x+  
20        " bob.x="+bob.x);  
21  
22    //printing objects implicitly calls toString()  
23    System.out.println("alice="+alice+  
24        " bob="+bob+" charlie="+charlie);  
25 }  
26 }
```

Stack Heap



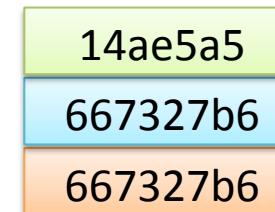
- Line 12, bob set equal to alice
- bob gets same value on stack that alice holds
- bob now points to the exact same memory location as alice
- bob and alice are “aliases” of each other

Declaring objects makes pointer on the stack, but object itself is elsewhere

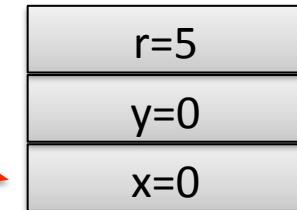
```
8 public static void main(String[] args) {  
9     //declare Blob objects  
10    Blob alice = new Blob();  
11    Blob bob; //notice no new keyword  
12    bob = alice; //bob equals alice  
13    Blob charlie = new Blob();  
14    System.out.println("alice.x="+alice.x+  
15                      " bob.x="+bob.x);  
16  
17    //update alice's x  
18    alice.setX(3);  
19    System.out.println("alice.x="+alice.x+  
20                      " bob.x="+bob.x);  
21  
22    //printing objects implicitly calls toString()  
23    System.out.println("alice="+alice+  
24                      " bob="+bob+" charlie="+charlie);  
25 }  
26 }
```

Stack

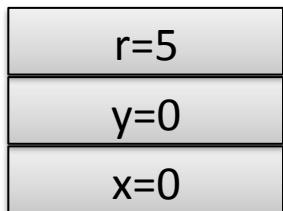
charlie
bob
alice



Heap



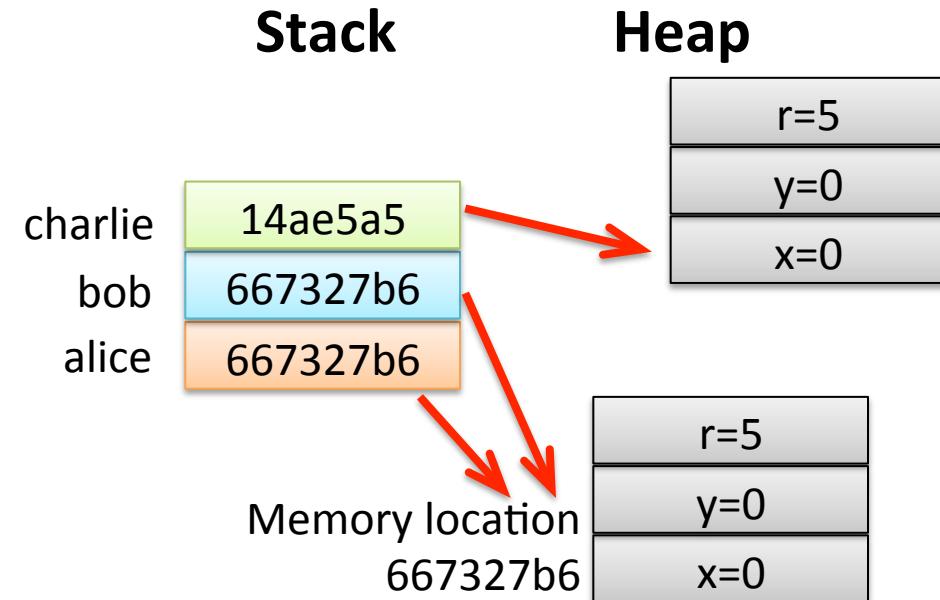
Memory location
667327b6



Charlie object gets new allocation elsewhere in memory because “new” keyword used

Declaring objects makes pointer on the stack, but object itself is elsewhere

```
8 public static void main(String[] args) {  
9     //declare Blob objects  
10    Blob alice = new Blob();  
11    Blob bob; //notice no new keyword  
12    bob = alice; //bob equals alice  
13    Blob charlie = new Blob();  
14    System.out.println("alice.x="+alice.x+  
15                      " bob.x="+bob.x);  
16  
17    //update alice's x  
18    alice.setX(3);  
19    System.out.println("alice.x="+alice.x+  
20                      " bob.x="+bob.x);  
21  
22    //printing objects implicitly calls toString()  
23    System.out.println("alice="+alice+  
24                      " bob="+bob+" charlie="+charlie);  
25 }  
26 }
```



x value for alice and bob is the same because stored at the exact same memory address

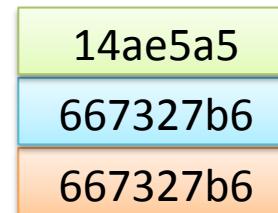
Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy
<terminated> MemoryAllocationObjects [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec
alice.x=0.0 bob.x=0.0

Declaring objects makes pointer on the stack, but object itself is elsewhere

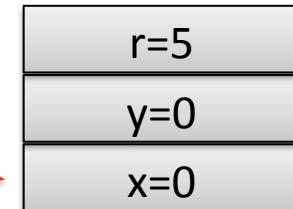
```
8 public static void main(String[] args) {  
9     //declare Blob objects  
10    Blob alice = new Blob();  
11    Blob bob; //notice no new keyword  
12    bob = alice; //bob equals alice  
13    Blob charlie = new Blob();  
14    System.out.println("alice.x="+alice.x+  
15                      " bob.x="+bob.x);  
16  
17    //update alice's x  
18    alice.setX(3);  
19    System.out.println("alice.x="+alice.x+  
20                      " bob.x="+bob.x);  
21  
22    //printing objects implicitly calls toString()  
23    System.out.println("alice="+alice+  
24                      " bob="+bob+" charlie="+charlie);  
25 }  
26 }
```

Stack

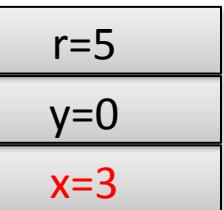
charlie
bob
alice



Heap



Memory location
667327b6



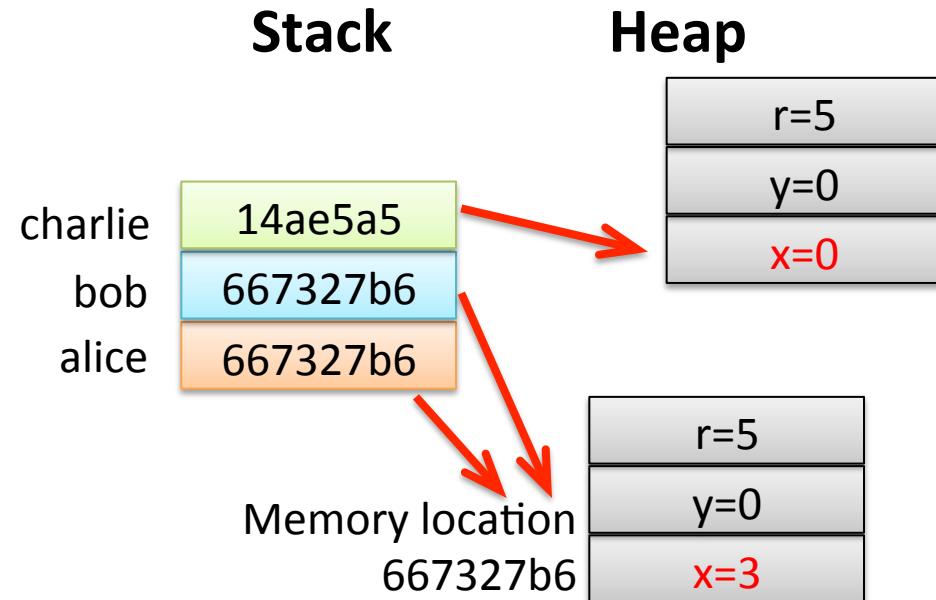
- **alice.x set to 3**
- **What is bob.x?**

Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy
<terminated> MemoryAllocationObjects [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec
alice.x=0.0 bob.x=0.0

Declaring objects makes pointer on the stack, but object itself is elsewhere

```
8 public static void main(String[] args) {  
9     //declare Blob objects  
10    Blob alice = new Blob();  
11    Blob bob; //notice no new keyword  
12    bob = alice; //bob equals alice  
13    Blob charlie = new Blob();  
14    System.out.println("alice.x="+alice.x+  
15                      " bob.x="+bob.x);  
16  
17    //update alice's x  
18    alice.setX(3);  
19    System.out.println("alice.x="+alice.x+  
20                      " bob.x="+bob.x);  
21  
22    //printing objects implicitly calls toString()  
23    System.out.println("alice="+alice+  
24                      " bob="+bob+" charlie="+charlie);  
25 }  
26 }
```

```
Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy  
<terminated> MemoryAllocationObjects [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec  
alice.x=0.0 bob.x=0.0  
alice.x=3.0 bob.x=3.0
```

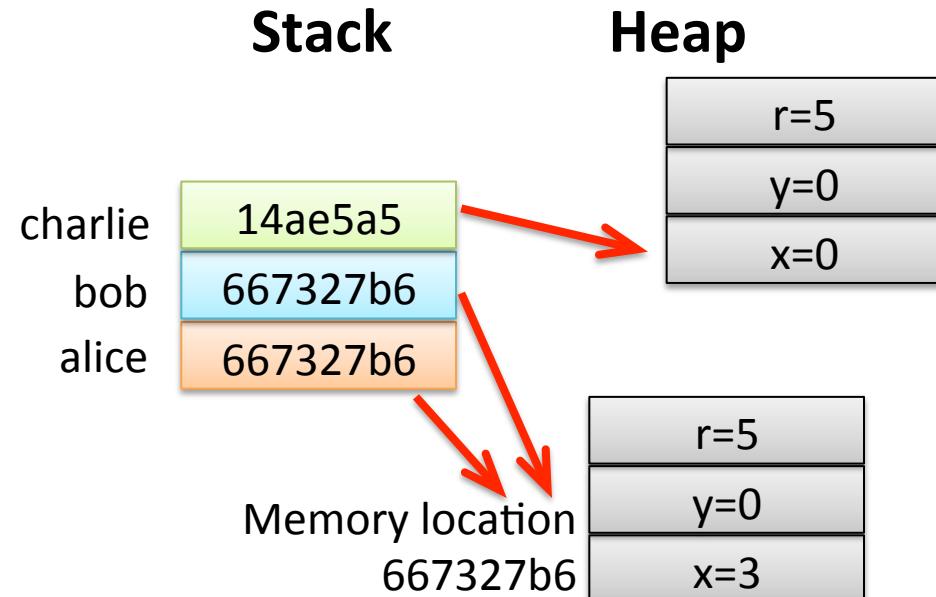


- **x is the same for both alice and bob objects because they point to the same memory address (called alias)**
- **Like Python setting two lists equal to each other, change one list, change the other also**
- **Charlie's x is still 0**

Declaring objects makes pointer on the stack, but object itself is elsewhere

```
8 public static void main(String[] args) {  
9     //declare Blob objects  
10    Blob alice = new Blob();  
11    Blob bob; //notice no new keyword  
12    bob = alice; //bob equals alice  
13    Blob charlie = new Blob();  
14    System.out.println("alice.x="+alice.x+  
15                      " bob.x="+bob.x);  
16  
17    //update alice's x  
18    alice.setX(3);  
19    System.out.println("alice.x="+alice.x+  
20                      " bob.x="+bob.x);  
21  
22    //printing objects implicitly calls toString()  
23    System.out.println("alice="+alice+  
24                      " bob="+bob+" charlie="+charlie);  
25 }  
26 }
```

```
Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy  
<terminated> MemoryAllocationObjects [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec  
alice.x=0.0 bob.x=0.0  
alice.x=3.0 bob.x=3.0  
alice=Blob@677327b6 bob=Blob@677327b6 charlie=Blob@14ae5a5
```



- Printing an object causes an implicit call to “`toString()`” function
- This can be overridden (see course webpage)
- By default `toString()` prints memory address of object (for primitives, value is printed)

Agenda

1. Objects vs. primitives
2. Inheritance
3. Quick review of Blobs
4. Using Inheritance to create different types of Blobs
5. Graphical User Interface (GUI) programming

Inheritance allows us to “build upon” work we have already done

Instance variables

Engines, fuel capacity,
top speed, stall speed

Base class
Airplane

Methods

Take off, land, climb,
descend, turn

May have already created a class called “Airplane”

- Defined instance variables common to all (or at least most) airplanes, such as “top speed” or “fuel capacity”
- Written methods for common actions such as “climb”

Inheritance allows us to use that work, plus extend it to define “specialty” versions of the base class (base class is also called super class; I’ll use both terms)

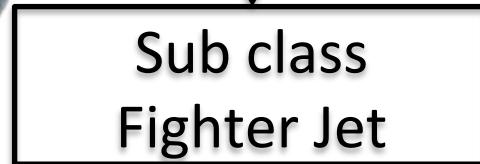
Subclasses get all the instance variables and methods of the base class (unless declared private in base class, then only the base class has access, not subclass)

Subclasses can also define new variables and methods, and can override the base class variables and methods with their own versions

Inheritance allows us to “build upon” work we have already done

Instance variables

Engines, fuel capacity,
top speed, stall speed



Instance variables

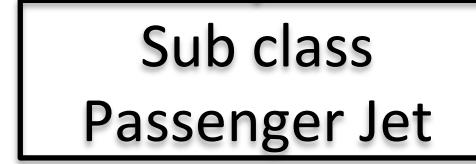
Weapons layout, ...

Methods

Take off, Fire weapon,...

Methods

Take off, land, climb,
descend, turn



Subclasses *inherit*
variables and
methods from
base class (no
need to rewrite)

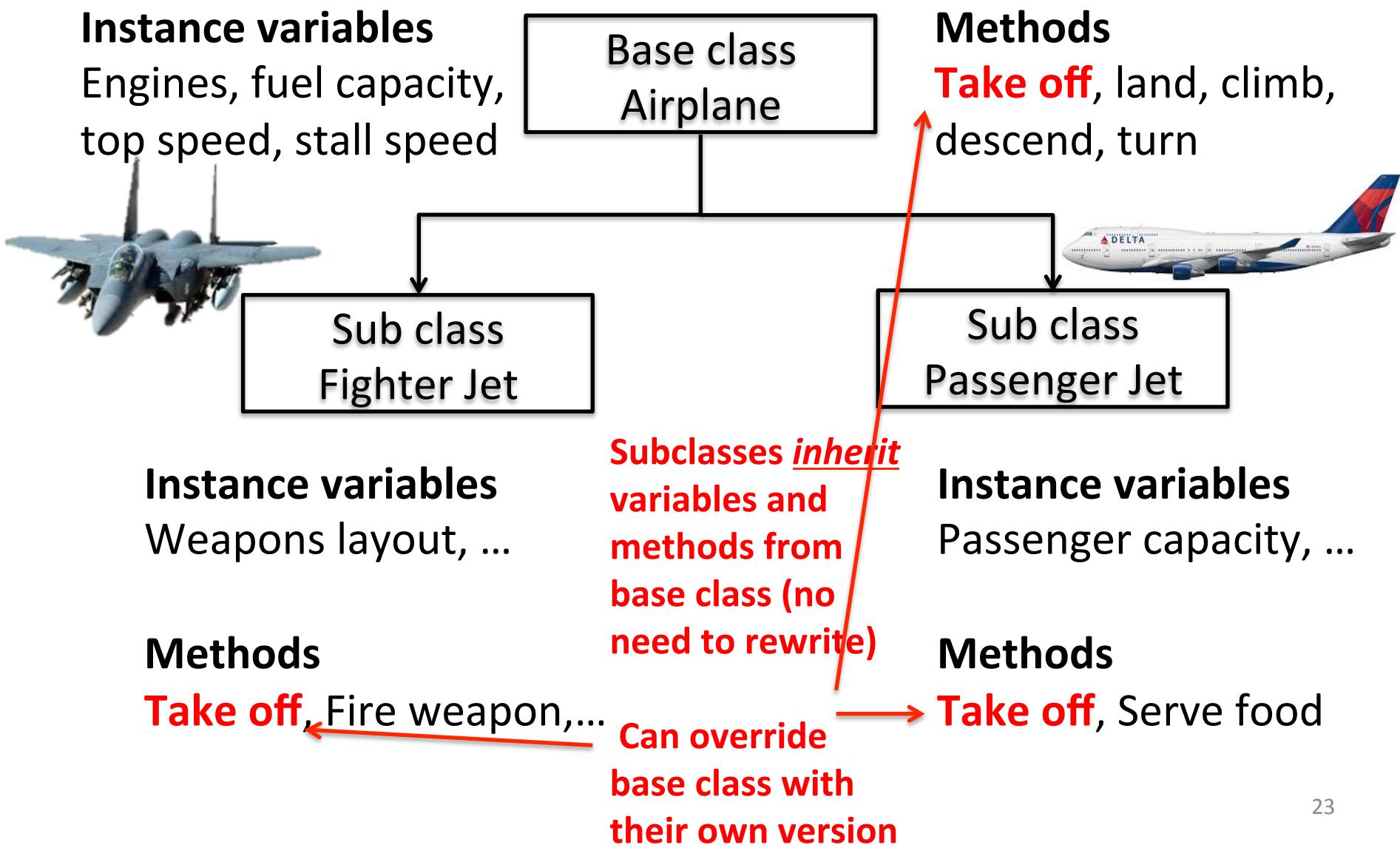
Instance variables

Passenger capacity, ...

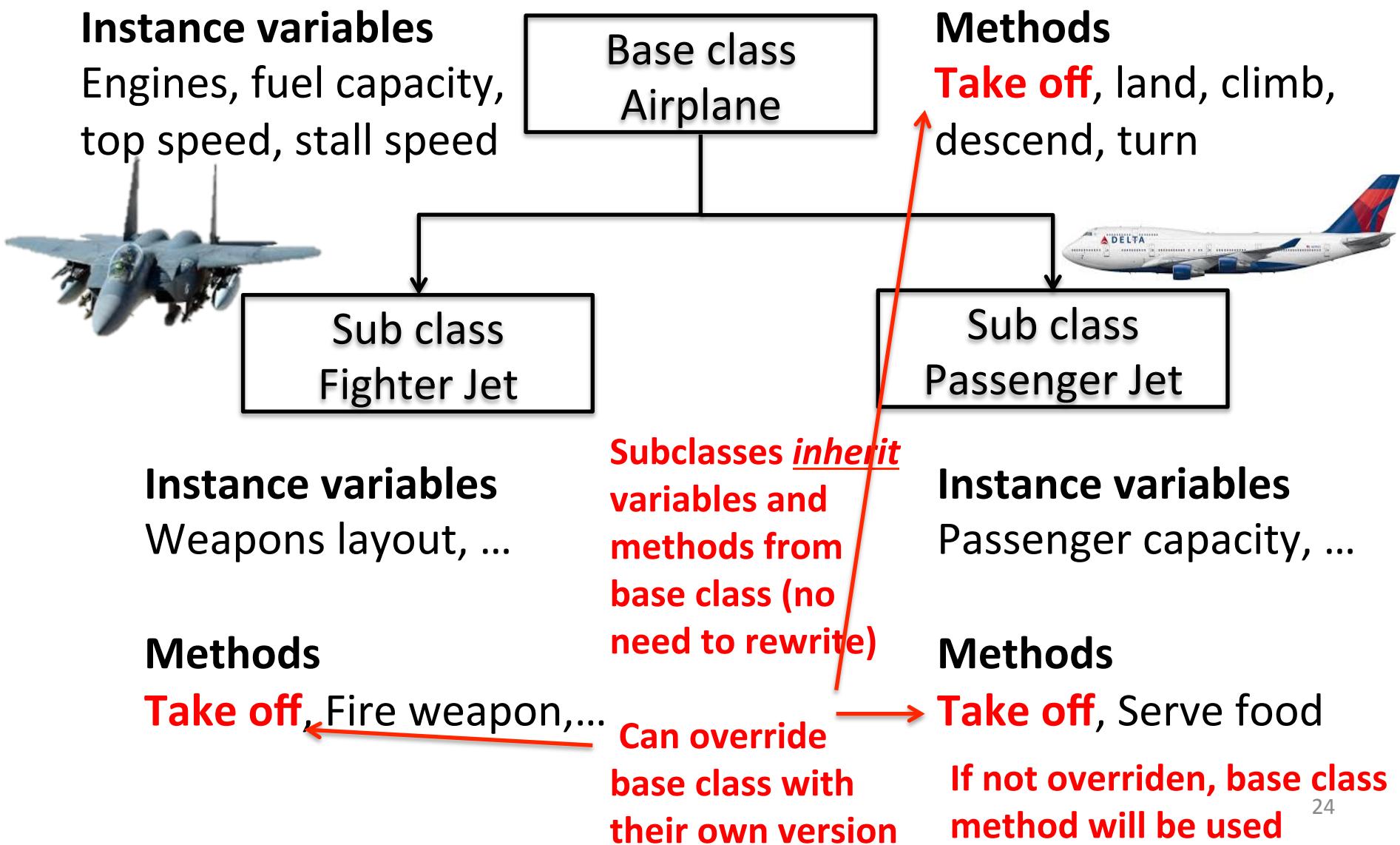
Methods

Take off, Serve food

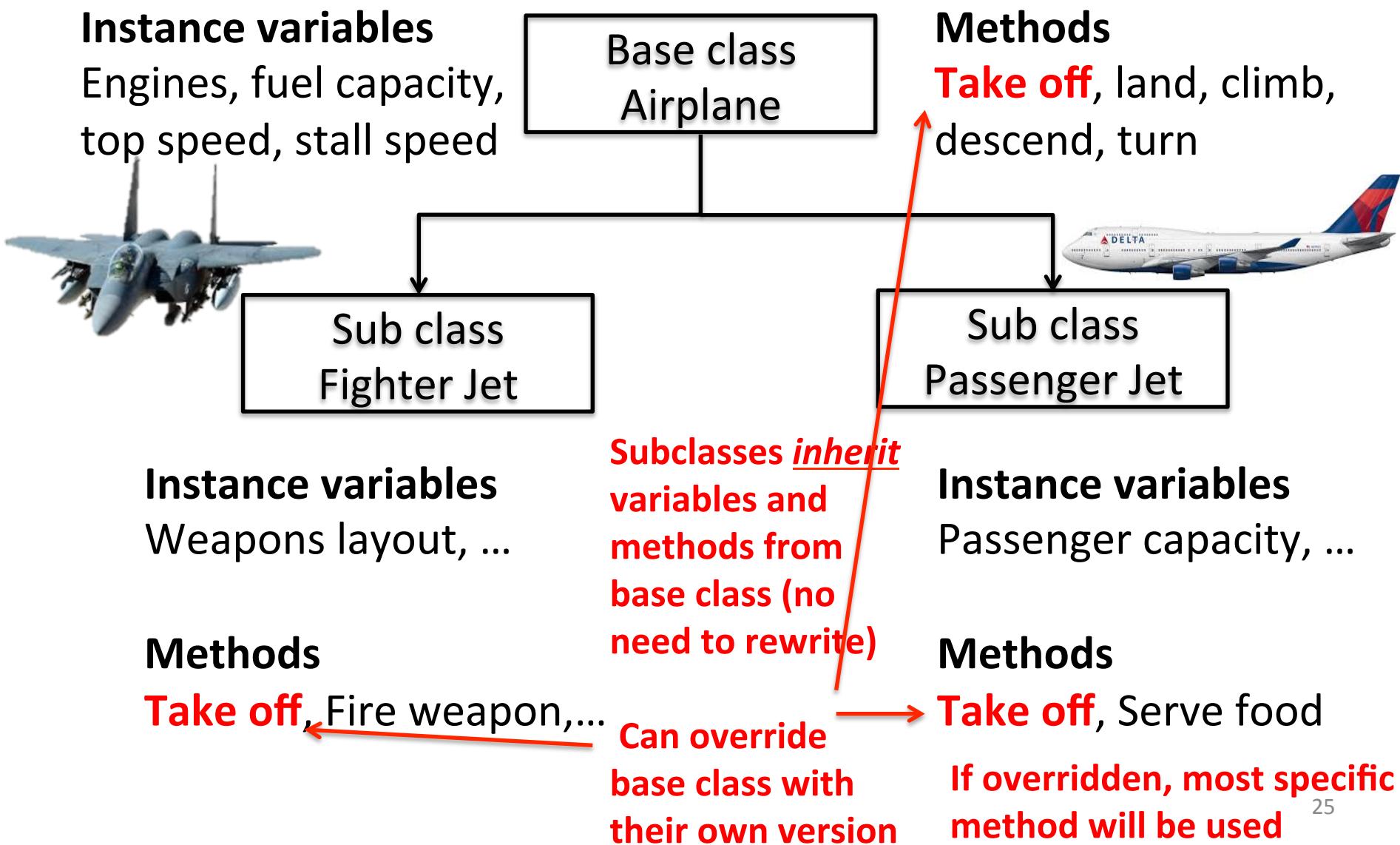
Inheritance allows us to “build upon” work we have already done



Inheritance allows us to “build upon” work we have already done



Inheritance allows us to “build upon” work we have already done



Inheritance models “is a” relationships

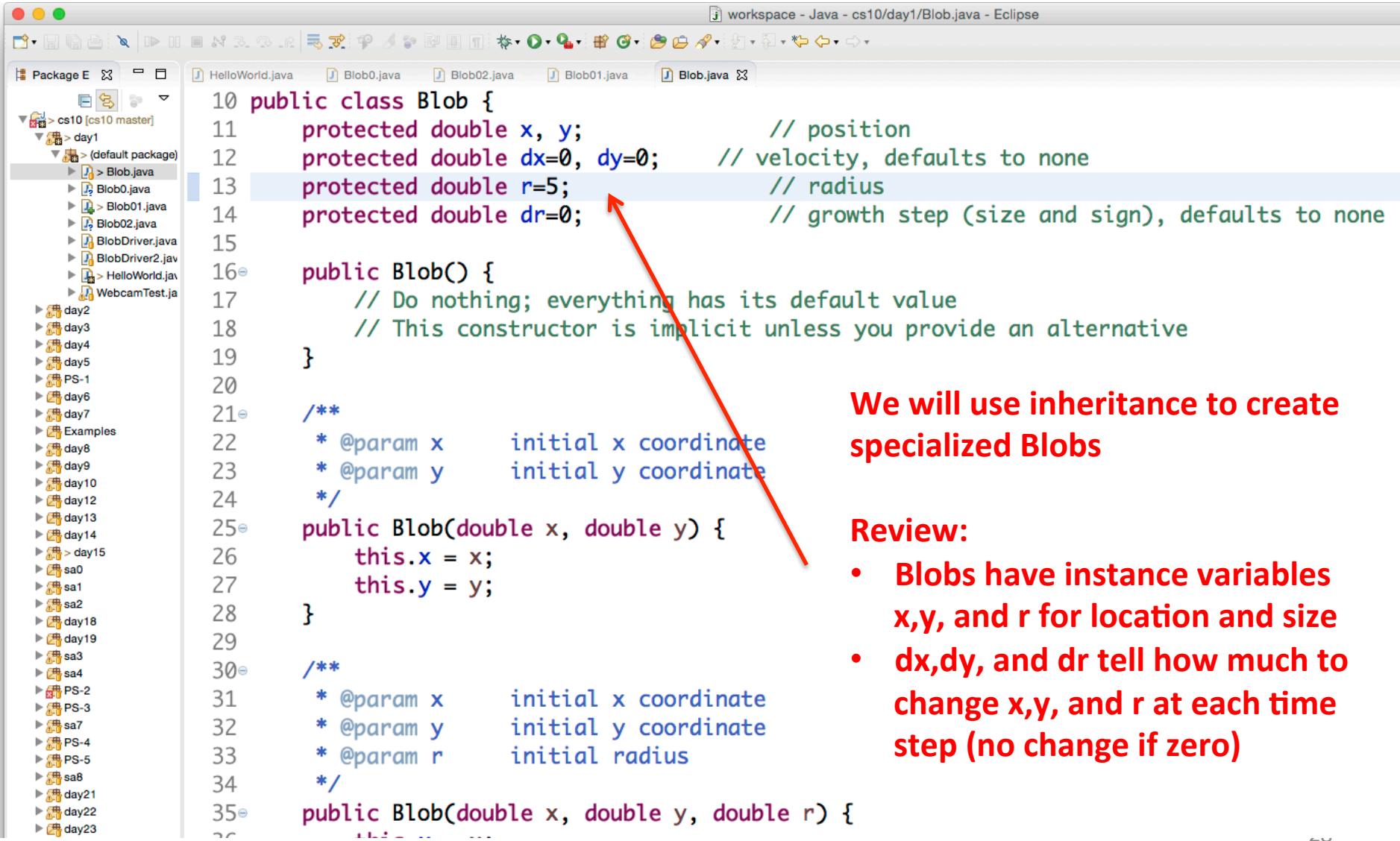
Inheritance rules

1. Every object of subclass “is a” object of base class
 - Every FighterJet “is a” Airplane
2. The set of objects of the subclass is a subset of the objects of base class
 - The set of FighterJets “is a” subset of Airplanes (there could be other types of airplanes)
 - Mathematically: $F \subseteq A$ where $F = \{\text{FighterJets}\}$, $A = \{\text{Airplanes}\}$
3. Every method that can be called on an object of base class, can also be called on an object of subclass
 - Results of the call may be very different if subclass overrides base class method, then subclass method is used
4. Objects of subclass are like objects of base class, but with additional specialization

Agenda

1. Objects vs. primitives
2. Inheritance
3. Quick review of Blobs
4. Using Inheritance to create different types of Blobs
5. Graphical User Interface (GUI) programming

Blob.java



The screenshot shows the Eclipse IDE interface with the title bar "workspace - Java - cs10/day1/Blob.java - Eclipse". The left sidebar displays a project structure for "cs10 [cs10 master]" containing various Java files and folders like "day1" through "day23", "Examples", and "PS-1" to "PS-5". The main editor window shows the Blob.java code:

```
10 public class Blob {  
11     protected double x, y;                      // position  
12     protected double dx=0, dy=0;                  // velocity, defaults to none  
13     protected double r=5;                        // radius  
14     protected double dr=0;                       // growth step (size and sign), defaults to none  
15  
16     public Blob() {  
17         // Do nothing; everything has its default value  
18         // This constructor is implicit unless you provide an alternative  
19     }  
20  
21     /**  
22      * @param x      initial x coordinate  
23      * @param y      initial y coordinate  
24      */  
25     public Blob(double x, double y) {  
26         this.x = x;  
27         this.y = y;  
28     }  
29  
30     /**  
31      * @param x      initial x coordinate  
32      * @param y      initial y coordinate  
33      * @param r      initial radius  
34      */  
35     public Blob(double x, double y, double r) {  
36         this.x = x;  
37         this.y = y;  
38         this.r = r;  
39     }  
40 }
```

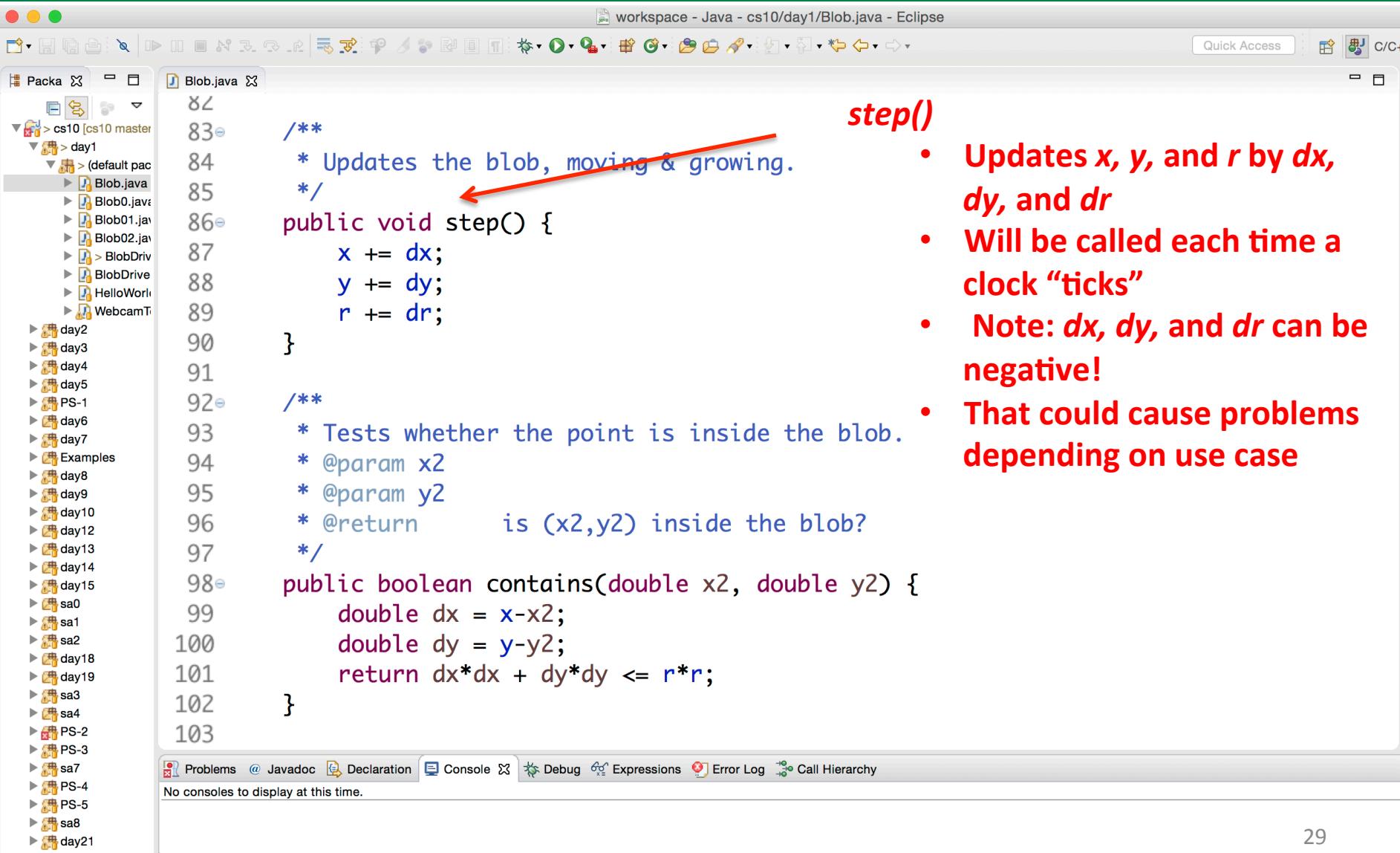
A red arrow points from the text "We will use inheritance to create specialized Blobs" down to the line "protected double r=5;" in the code.

We will use inheritance to create specialized Blobs

Review:

- **Blobs have instance variables x,y, and r for location and size**
- **dx,dy, and dr tell how much to change x,y, and r at each time step (no change if zero)**

Blob: New methods to control how the blob behaves



The screenshot shows the Eclipse IDE interface with the following details:

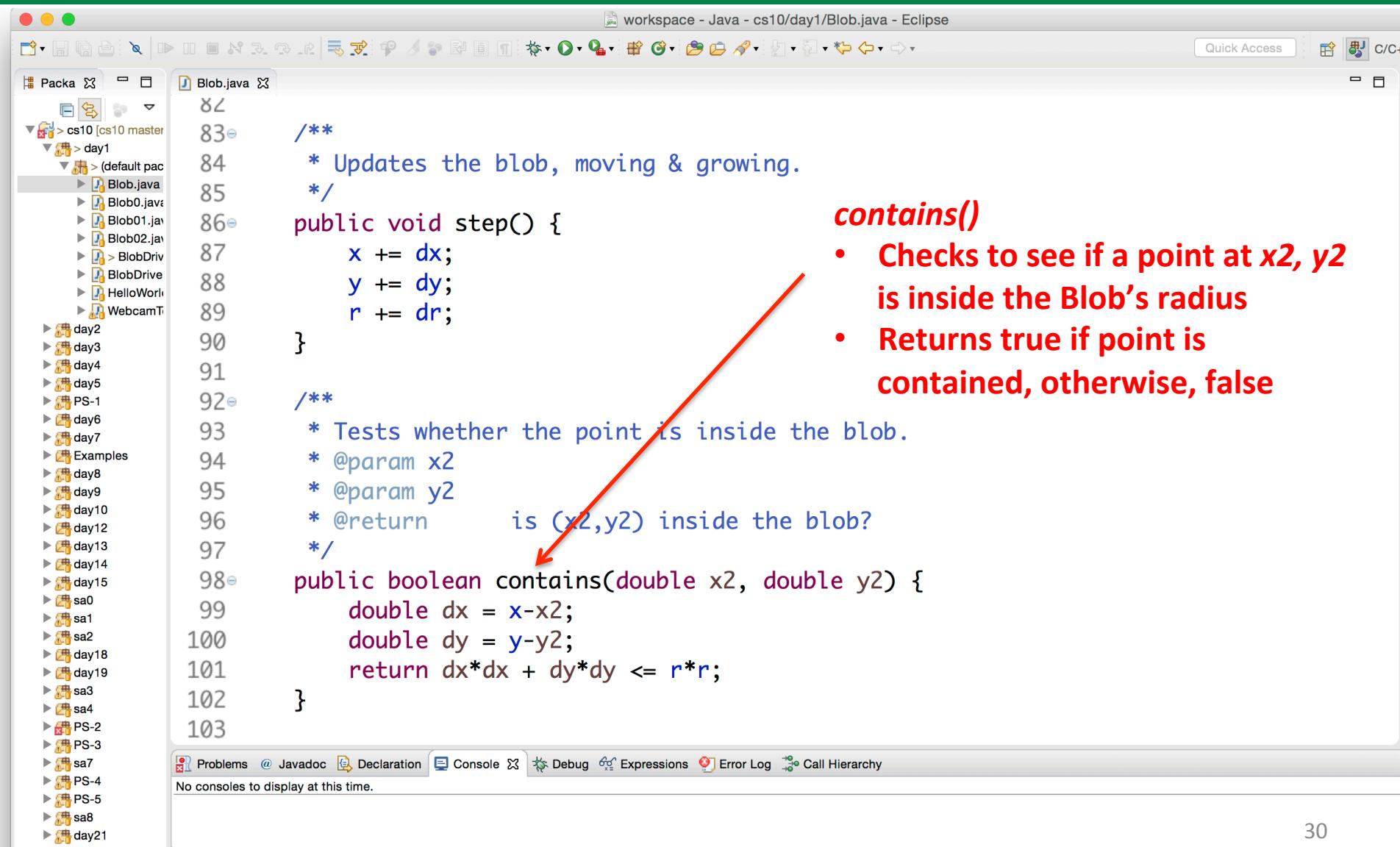
- Title Bar:** workspace - Java - cs10/day1/Blob.java - Eclipse
- Toolbar:** Standard Eclipse toolbar with various icons for file operations, search, and navigation.
- Left Sidebar:** Project Explorer showing the project structure under "cs10 [cs10 master]". The "day1" folder contains "Blob.java", "Blob0.java", "Blob01.java", "Blob02.java", "BlobDrive", and "HelloWorld". Other folders like "day2" through "day21" and "PS-1" through "PS-5" are also listed.
- Central Area:** Editor view showing the content of "Blob.java".

```
82
83  /**
84   * Updates the blob, moving & growing.
85   */
86  public void step() {
87      x += dx;
88      y += dy;
89      r += dr;
90  }
91
92  /**
93   * Tests whether the point is inside the blob.
94   * @param x2
95   * @param y2
96   * @return      is (x2,y2) inside the blob?
97   */
98  public boolean contains(double x2, double y2) {
99      double dx = x-x2;
100     double dy = y-y2;
101     return dx*dx + dy*dy <= r*r;
102 }
```
- Annotations:** A red arrow points from the word "step()" in the list below to the method definition in the code.
- Right Side:** A red box highlights the word "step()", which is followed by a list of bullet points describing its behavior.
- Bottom:** Eclipse status bar showing tabs for Problems, Javadoc, Declaration, Console, Debug, Expressions, Error Log, and Call Hierarchy. It also says "No consoles to display at this time."

step()

- **Updates x, y, and r by dx, dy, and dr**
- **Will be called each time a clock “ticks”**
- **Note: dx, dy, and dr can be negative!**
- **That could cause problems depending on use case**

Blob: New methods to control how the blob behaves



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows the workspace structure under "cs10 [cs10 master]". The "day1" folder contains "Blob.java", "Blob0.java", "Blob01.java", "Blob02.java", "BlobDrive", "HelloWorld", and "WebcamT". Other folders like "day2" through "day21" and "PS-1" through "PS-5" are also listed.
- Code Editor (right):** Displays the content of "Blob.java".

```
82
83     /**
84      * Updates the blob, moving & growing.
85      */
86     public void step() {
87         x += dx;
88         y += dy;
89         r += dr;
90     }
91
92     /**
93      * Tests whether the point is inside the blob.
94      * @param x2
95      * @param y2
96      * @return      is (x2,y2) inside the blob?
97      */
98     public boolean contains(double x2, double y2) {
99         double dx = x-x2;
100        double dy = y-y2;
101        return dx*dx + dy*dy <= r*r;
102    }
103}
```
- Annotations:** A red arrow points from the explanatory text "is (x2,y2) inside the blob?" down to the `contains` method definition.
- Toolbars and Status Bar:** Standard Eclipse toolbars and status bar at the bottom.

contains()

- Checks to see if a point at x_2, y_2 is inside the Blob's radius
- Returns true if point is contained, otherwise, false

Blob: We did not discuss the *draw()* function yesterday

```
► day12
► day13
► day14
► > day15
► sa0
► sa1
► sa2
► day18
► day19
► sa3
► sa4
► PS-2
► PS-3
► sa7
► PS-4
► PS-5
► sa8
► day21
► day22
► day23
► PS-6
► midterm1
► day24
► mazeSearch
► midterm2
► > final
► JRE System Library [ ]
► Referenced Libraries
► data
► inputs
► pictures
► solns
```

```
96     * @return      is (x2,y2) inside the blob?
97     */
98     public boolean contains(double x2, double y2) {
99         double dx = x-x2;
100        double dy = y-y2;
101        return dx*dx + dy*dy <= r*r; draw()
102    }
103
104    /**
105     * Draws the blob on the graphics. Called by java.awt everytime repaint called.
106     * @param g
107     */
108    public void draw(Graphics g) {
109        g.fillOval((int)(x-r), (int)(y-r), (int)(2*r), (int)(2*r));
110    }
111 }
```

- Used to display Blob on screen

fillOval()

- Actually draws Blob

- Integer params, cast double to int
- Params: left most, top most, size x, size y

Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

No consoles to display at this time.

Blob: We did not discuss the draw() function last class

```
► day12  
► day13  
► day14  
► > day15  
► sa0  
► sa1  
► sa2  
► day18  
► day19  
► sa3  
► sa4  
► PS-2  
► PS-3  
► sa7  
► PS-4  
► PS-5  
► sa8  
► day21  
► day22  
► day23  
► PS-6  
► midterm1  
► day24  
► mazeSearch  
► midterm2  
► > final  
► JRE System Library [ ]  
► Referenced Libraries  
► data  
► inputs  
► pictures  
► solns
```

```
96     * @return      is (x2,y2) inside the blob?  
97     */  
98     public boolean contains(double x2, double y2) {  
99         double dx = x-x2;  
100        double dy = y-y2;  
101        return dx*dx + dy*dy <= r*r;  
102    }  
103  
104    /**  
105     * Draws the blob on the graphics. Called by paint everytime repaint called.  
106     * @param g  
107     */  
108    public void draw(Graphics g) {  
109        g.fillOval((int)(x-r), (int)(y-r), (int)(2*r), (int)(2*r));  
110    }  
111 }
```

draw()

- Used to display Blob on screen



fillOval()

- Actually draws Blob



Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

No consoles to display at this time.

- Integer params, cast double to int
- Params: left most, top most, width, height
- This math centers blob at x,y

Agenda

1. Objects vs. primitives
2. Inheritance
3. Quick review of Blobs
4. Using Inheritance to create different types of Blobs
5. Graphical User Interface (GUI) programming

Use inheritance to create specialized Blobs that have different behavior

Instance variables

- x, y, r
- dx, dy, dr

Blob

Methods

- *getters/setters*
- *step()*
- *contains()*
- *draw()*

When *step()* called, blob moves location by dx, dy and changes size by dr

We can create “specialty” blobs that behave differently from the base blob when *step()* called

Specialty blobs (called subclasses) inherit the base blob’s instance variables and methods (so no need to re-write them!)

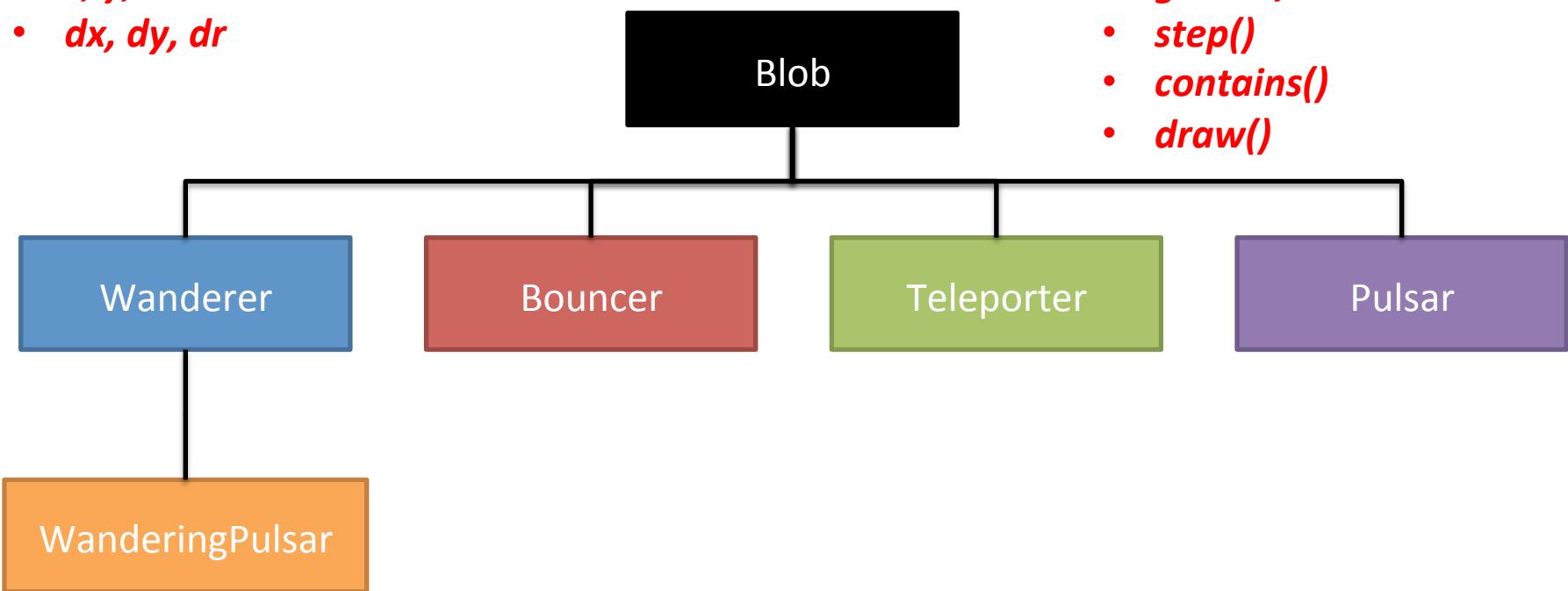
Use inheritance to create specialized Blobs that have different behavior

Instance variables

- x, y, r
- dx, dy, dr

Methods

- *getters/setters*
- *step()*
- *contains()*
- *draw()*



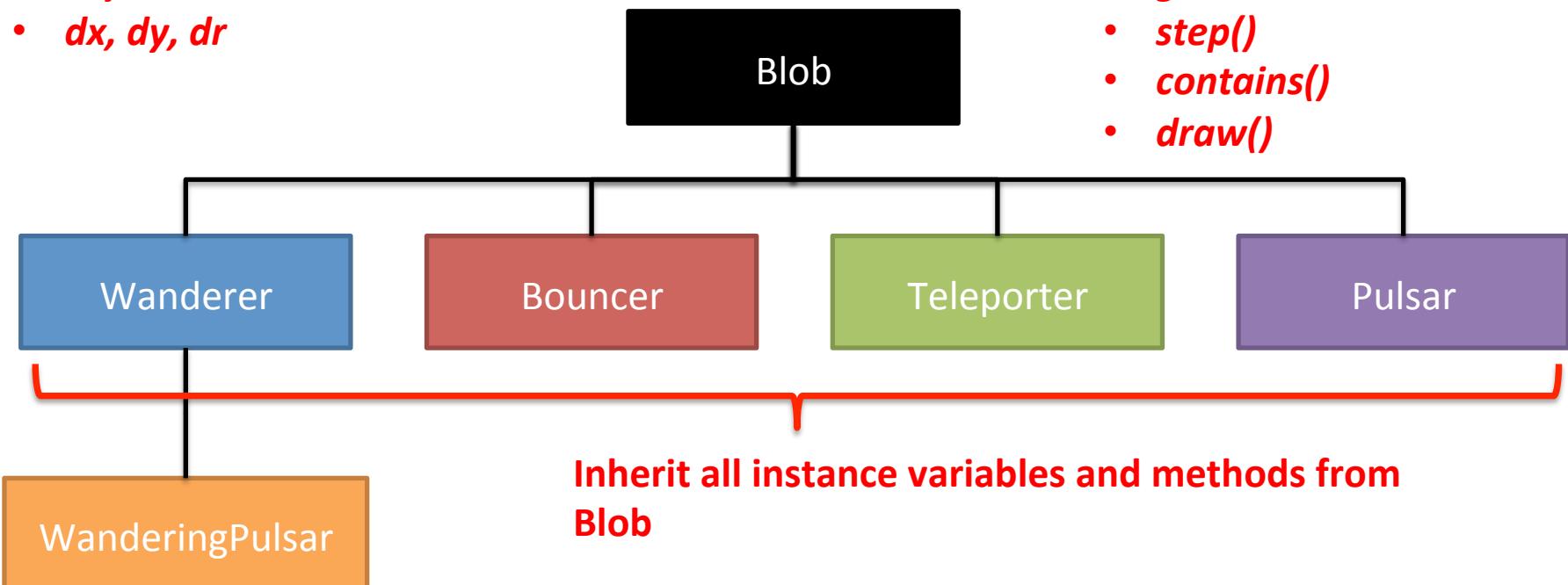
Use inheritance to create specialized Blobs that have different behavior

Instance variables

- *x, y, r*
- *dx, dy, dr*

Methods

- *getters/setters*
- *step()*
- *contains()*
- *draw()*



Inherit all instance variables and methods from Blob

Can override base class methods with “specialty” versions in subclass (this is Polymorphism)

If subclass defines method, subclass’s method is used, otherwise base class method used (dynamic dispatch)

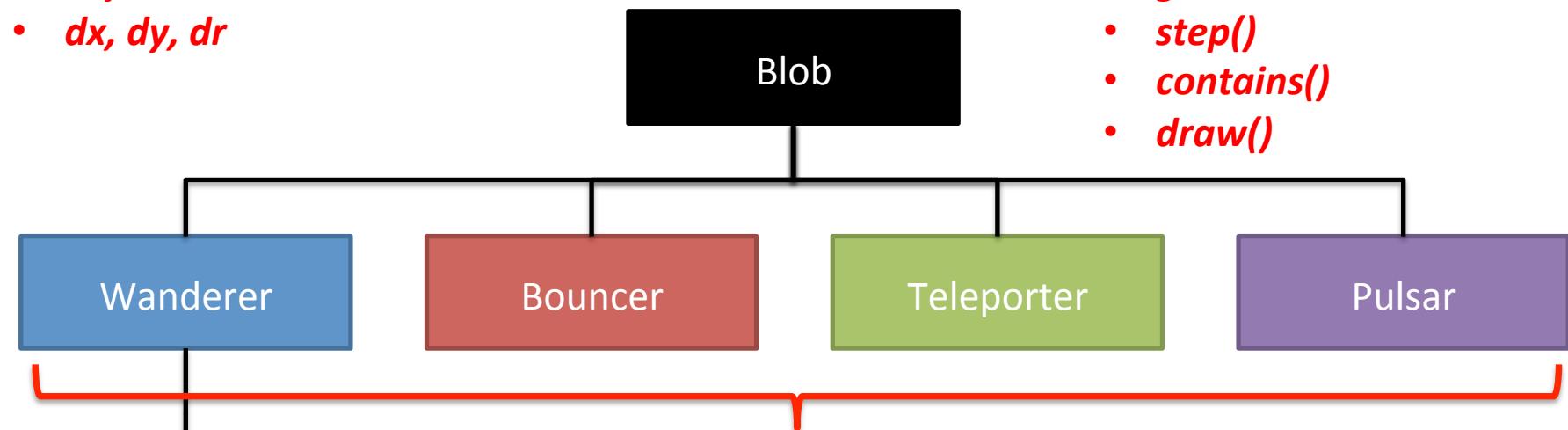
Use inheritance to create specialized Blobs that have different behavior

Instance variables

- *x, y, r*
- *dx, dy, dr*

Methods

- *getters/setters*
- *step()*
- *contains()*
- *draw()*



Inherit all instance variables and methods from Blob

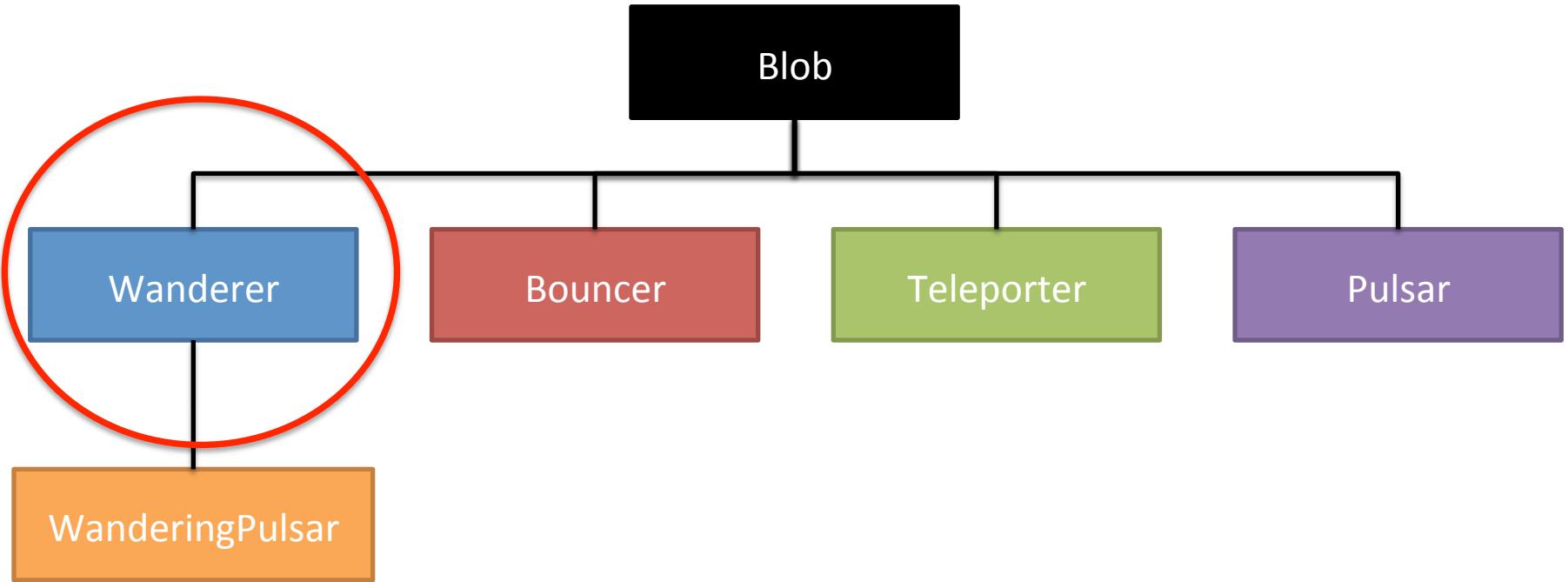
Can override base class methods with “specialty” versions in subclass (this is Polymorphism)

If subclass defines method, subclass’s method is used, otherwise base class method used (dynamic dispatch)

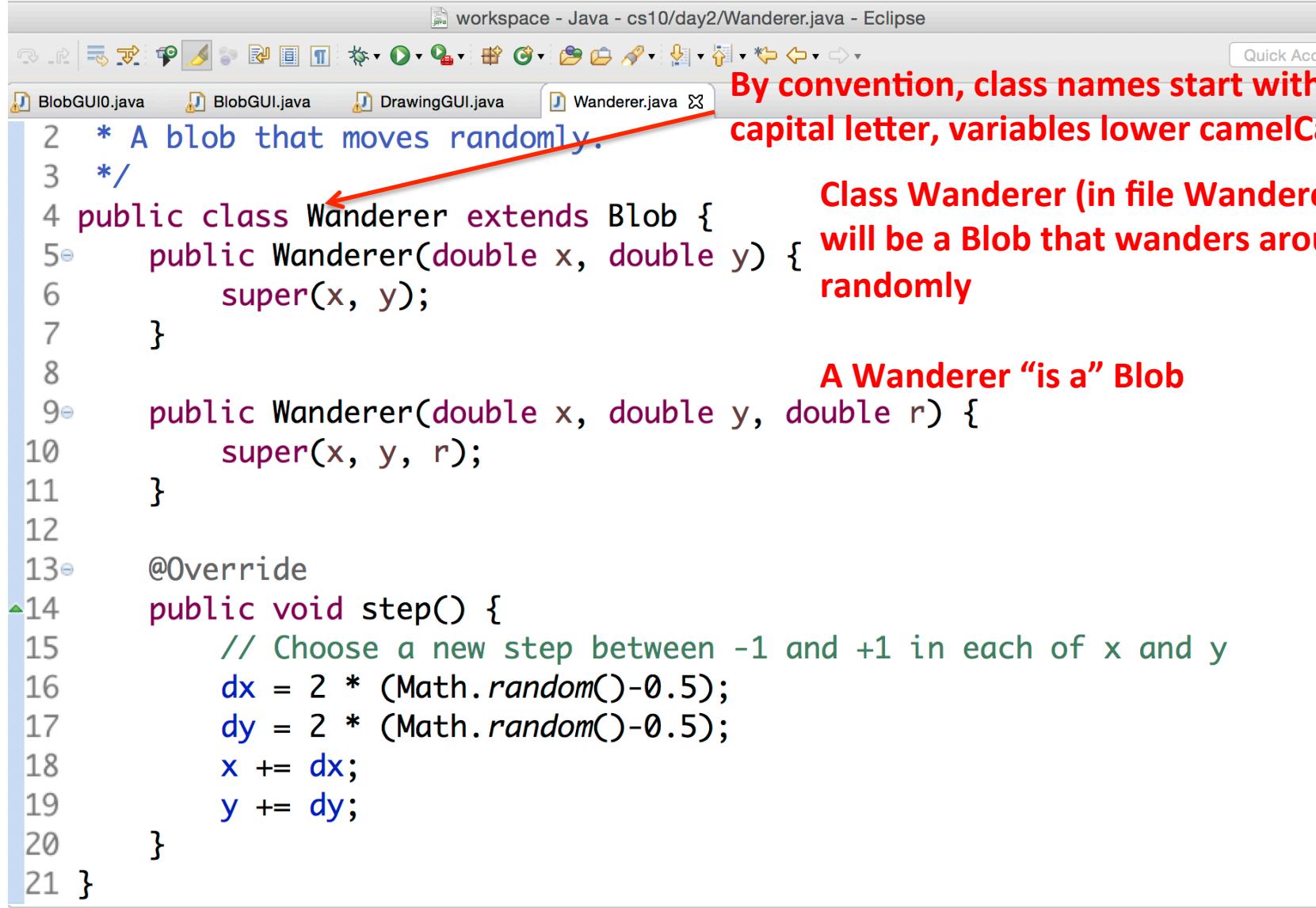
Inherits instance variables and methods from Wanderer

In Java, cannot inherit from multiple base classes

A Wanderer will move around randomly



A Wanderer “is a” type of Blob that moves around randomly



The screenshot shows the Eclipse IDE interface with the title bar "workspace - Java - cs10/day2/Wanderer.java - Eclipse". The toolbar has various icons for file operations. Below the toolbar, there are four tabs: "BlobGUI0.java", "BlobGUI.java", "DrawingGUI.java", and "Wanderer.java". The "Wanderer.java" tab is selected. The code editor displays the following Java code:

```
2 * A blob that moves randomly.
3 */
4 public class Wanderer extends Blob {
5     public Wanderer(double x, double y) {
6         super(x, y);
7     }
8
9     public Wanderer(double x, double y, double r) {
10        super(x, y, r);
11    }
12
13    @Override
14    public void step() {
15        // Choose a new step between -1 and +1 in each of x and y
16        dx = 2 * (Math.random() - 0.5);
17        dy = 2 * (Math.random() - 0.5);
18        x += dx;
19        y += dy;
20    }
21 }
```

A red arrow points from the text "By convention, class names start with capital letter, variables lower camelCase" to the word "Wanderer" in the class definition.

By convention, class names start with capital letter, variables lower camelCase

Class Wanderer (in file Wanderer.java) will be a Blob that wanders around randomly

A Wanderer “is a” Blob

A Wanderer “is a” type of Blob that moves around randomly

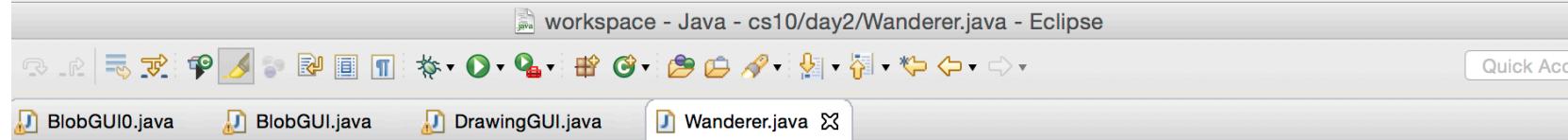
The screenshot shows the Eclipse IDE interface with the title bar "workspace - Java - cs10/day2/Wanderer.java - Eclipse". Below the title bar is a toolbar with various icons. The main area displays the code for `Wanderer.java`. The code defines a class `Wanderer` that extends the `Blob` class. It includes two constructors and an overridden `step()` method.

```
2 * A blob that moves randomly.  
3 */  
4 public class Wanderer extends Blob {  
5     public Wanderer(double x, double y) {  
6         super(x, y);  
7     }  
8  
9     public Wanderer(double x, double y, double r) {  
10        super(x, y, r);  
11    }  
12  
13    @Override  
14    public void step() {  
15        // Choose a new step between -1 and +1 in each of x and y  
16        dx = 2 * (Math.random() - 0.5);  
17        dy = 2 * (Math.random() - 0.5);  
18        x += dx;  
19        y += dy;  
20    }  
21 }
```

Annotations in red text are overlaid on the code:

- "By convention, class names start with capital letter, variables lower camelCase" points to the `Wanderer` class name.
- "Class Wanderer (in file Wanderer.java) will be a Blob that wanders around randomly" points to the `extends Blob` line.
- "A Wanderer “is a” Blob" points to the `Wanderer` class definition.
- "Java keyword “extends” means inherit instance variables and methods from the base class that follows “extends” (Blob here)" points to the `extends Blob` line.
- "Wanderer inherits from Blob" points to the `super` calls in the constructors.
- "Wanderer “is a” subclass of Blob" points to the `super` calls in the constructors.
- "Wanderer has all the instance variables and methods defined in Blob super class (aka base class)" points to the `step()` method implementation.

A Wanderer “is a” type of Blob that moves around randomly



```
2 * A blob that moves randomly.  
3 */  
4 public class Wanderer extends Blob {  
5     public Wanderer(double x, double y) {  
6         super(x, y);  
7     }  
8  
9     public Wanderer(double x, double y, double r) {  
10        super(x, y, r);  
11    }  
12  
13    @Override  
14    public void step() {  
15        // Choose a new step between -1 and +1 in each of x and y  
16        dx = 2 * (Math.random() - 0.5);  
17        dy = 2 * (Math.random() - 0.5);  
18        x += dx;  
19        y += dy;  
20    }  
21 }
```

Wanderer constructors call “super()” method

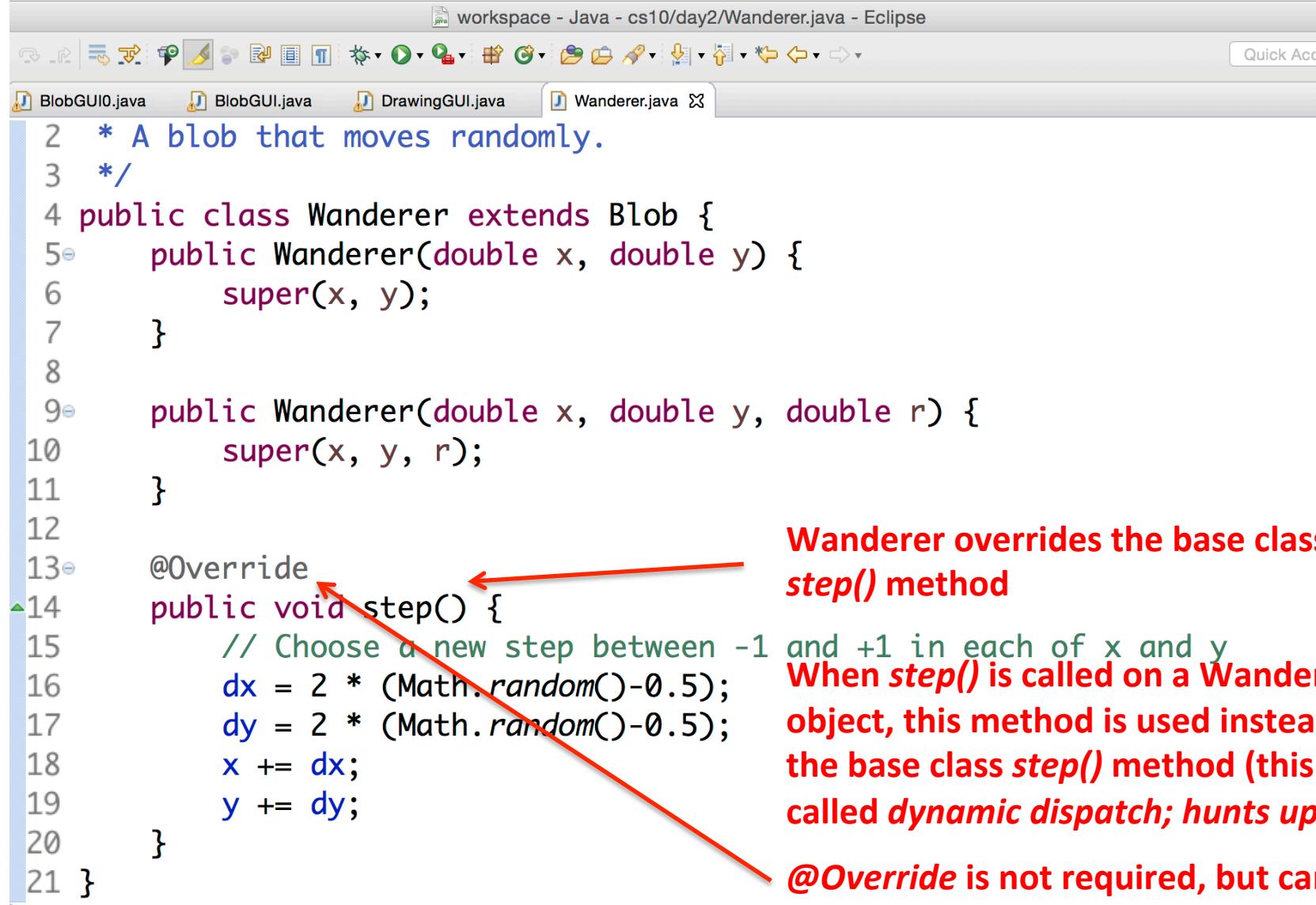
super() is the super class (aka base class) constructor

No need to rewrite the constructor code if we aren’t going to do anything different – just reuse the existing constructor code by calling super()

Works on methods other than just constructors (but constructors don’t have a name so identify with “super”)

Could call setX() and it would call base class setX()

A Wanderer “is a” type of Blob that moves around randomly



The screenshot shows the Eclipse IDE interface with the title bar "workspace - Java - cs10/day2/Wanderer.java - Eclipse". The toolbar has various icons for file operations. Below the toolbar, there are tabs for "BlobGUI0.java", "BlobGUI.java", "DrawingGUI.java", and "Wanderer.java", with "Wanderer.java" being the active tab. The code editor displays the following Java code:

```
2 * A blob that moves randomly.  
3 */  
4 public class Wanderer extends Blob {  
5     public Wanderer(double x, double y) {  
6         super(x, y);  
7     }  
8  
9     public Wanderer(double x, double y, double r) {  
10        super(x, y, r);  
11    }  
12  
13    @Override  
14    public void step() {  
15        // Choose a new step between -1 and +1 in each of x and y  
16        dx = 2 * (Math.random() - 0.5);  
17        dy = 2 * (Math.random() - 0.5);  
18        x += dx;  
19        y += dy;  
20    }  
21 }
```

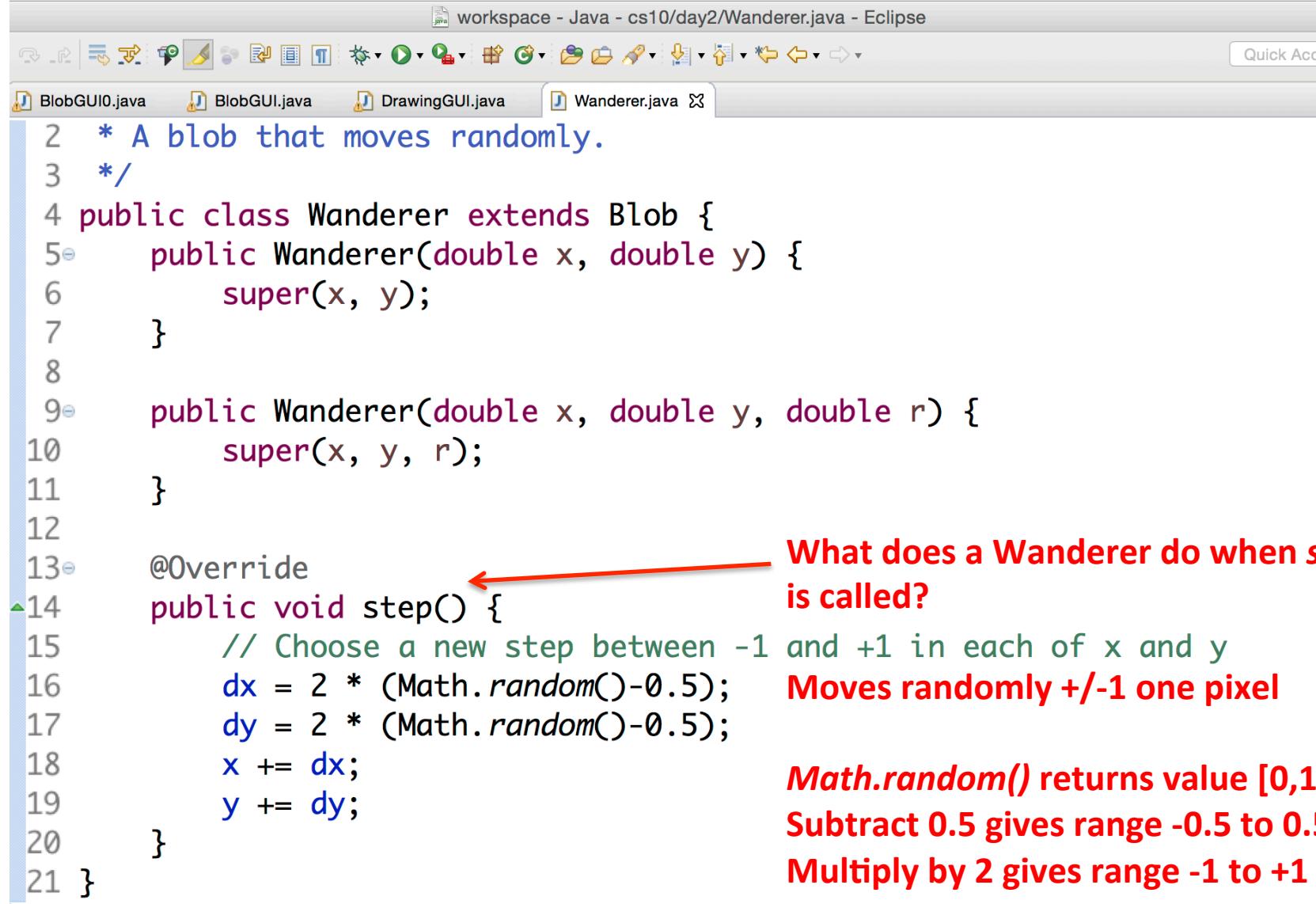
A red arrow points from the text "Wanderer overrides the base class *step()* method" to the `@Override` annotation on line 13. Another red arrow points from the text "@Override is not required, but can help prevent errors" to the `@Override` annotation on line 13.

Wanderer overrides the base class *step()* method

When *step()* is called on a Wanderer object, this method is used instead of the base class *step()* method (this is called *dynamic dispatch; hunts upward*)

@Override is not required, but can help prevent errors

A Wanderer “is a” type of Blob that moves around randomly



The screenshot shows the Eclipse IDE interface with the title bar "workspace - Java - cs10/day2/Wanderer.java - Eclipse". The toolbar has various icons for file operations. Below the toolbar, there are tabs for "BlobGUI0.java", "BlobGUI.java", "DrawingGUI.java", and "Wanderer.java", with "Wanderer.java" currently selected. The code editor displays the following Java code:

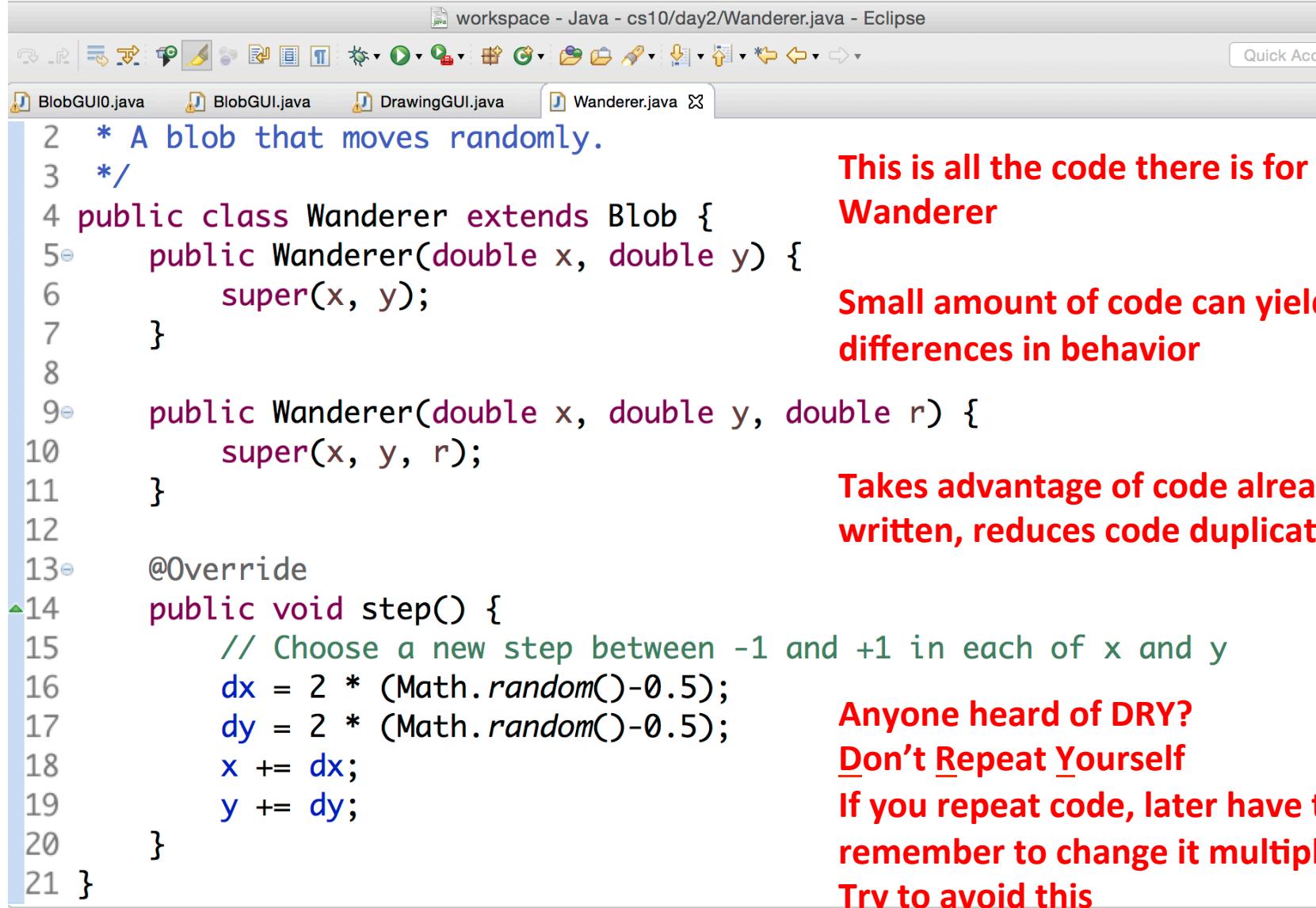
```
2 * A blob that moves randomly.  
3 */  
4 public class Wanderer extends Blob {  
5     public Wanderer(double x, double y) {  
6         super(x, y);  
7     }  
8  
9     public Wanderer(double x, double y, double r) {  
10        super(x, y, r);  
11    }  
12  
13    @Override  
14    public void step() {  
15        // Choose a new step between -1 and +1 in each of x and y  
16        dx = 2 * (Math.random() - 0.5);  
17        dy = 2 * (Math.random() - 0.5);  
18        x += dx;  
19        y += dy;  
20    }  
21 }
```

A red arrow points from the question "What does a Wanderer do when `step()` is called?" to the `step()` method definition.

What does a Wanderer do when `step()` is called?

`Math.random()` returns value [0,1)
Subtract 0.5 gives range -0.5 to 0.5
Multiply by 2 gives range -1 to +1

A Wanderer “is a” type of Blob that moves around randomly



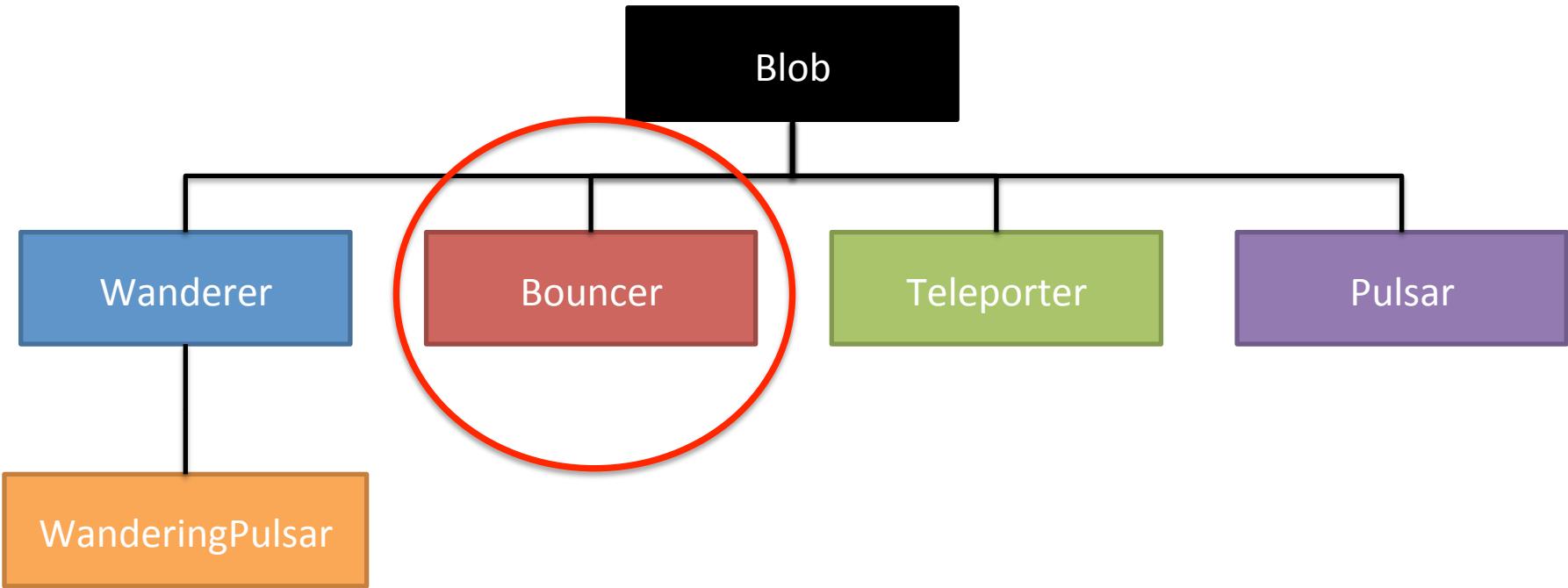
The screenshot shows the Eclipse IDE interface with the title bar "workspace - Java - cs10/day2/Wanderer.java - Eclipse". The toolbar has various icons for file operations like new, open, save, and run. Below the toolbar is a tab bar with four tabs: "BlobGUI0.java", "BlobGUI.java", "DrawingGUI.java", and "Wanderer.java" (which is currently selected). The code editor displays the following Java code:

```
2 * A blob that moves randomly.  
3 */  
4 public class Wanderer extends Blob {  
5     public Wanderer(double x, double y) {  
6         super(x, y);  
7     }  
8  
9     public Wanderer(double x, double y, double r) {  
10        super(x, y, r);  
11    }  
12  
13    @Override  
14    public void step() {  
15        // Choose a new step between -1 and +1 in each of x and y  
16        dx = 2 * (Math.random() - 0.5);  
17        dy = 2 * (Math.random() - 0.5);  
18        x += dx;  
19        y += dy;  
20    }  
21 }
```

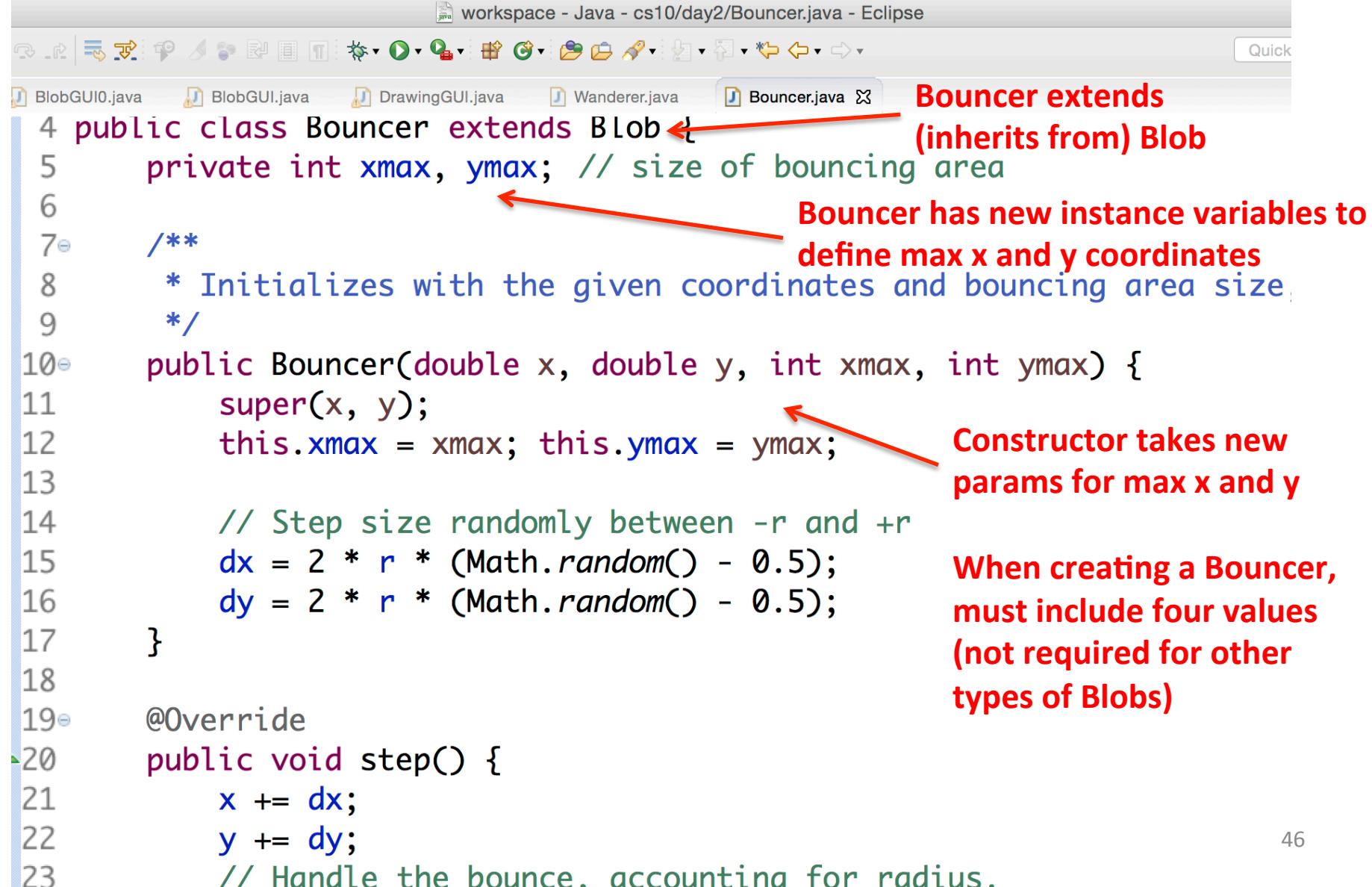
Annotations in red text are placed next to the code:

- "This is all the code there is for a Wanderer"
- "Small amount of code can yield big differences in behavior"
- "Takes advantage of code already written, reduces code duplication"
- "Anyone heard of DRY?
Don't Repeat Yourself
If you repeat code, later have to remember to change it multiple places
Try to avoid this"

A Bouncer will move in a straight line until it hits a wall, then it will rebound



A Bouncer “is a” type of Blob that rebounds off sides



The screenshot shows the Eclipse IDE interface with the title bar "workspace - Java - cs10/day2/Bouncer.java - Eclipse". Below the title bar is the toolbar with various icons. The main area displays the code for Bouncer.java, which extends Blob. The code includes instance variables for max coordinates and bouncing area size, and a constructor that takes parameters for these values. It also overrides the step() method to handle bouncing logic.

```
4 public class Bouncer extends Blob {  
5     private int xmax, ymax; // size of bouncing area  
6  
7     /**  
8      * Initializes with the given coordinates and bouncing area size.  
9     */  
10    public Bouncer(double x, double y, int xmax, int ymax) {  
11        super(x, y);  
12        this.xmax = xmax; this.ymax = ymax;  
13  
14        // Step size randomly between -r and +r  
15        dx = 2 * r * (Math.random() - 0.5);  
16        dy = 2 * r * (Math.random() - 0.5);  
17    }  
18  
19    @Override  
20    public void step() {  
21        x += dx;  
22        y += dy;  
23        // Handle the bounce, accounting for radius.  
24    }  
25}
```

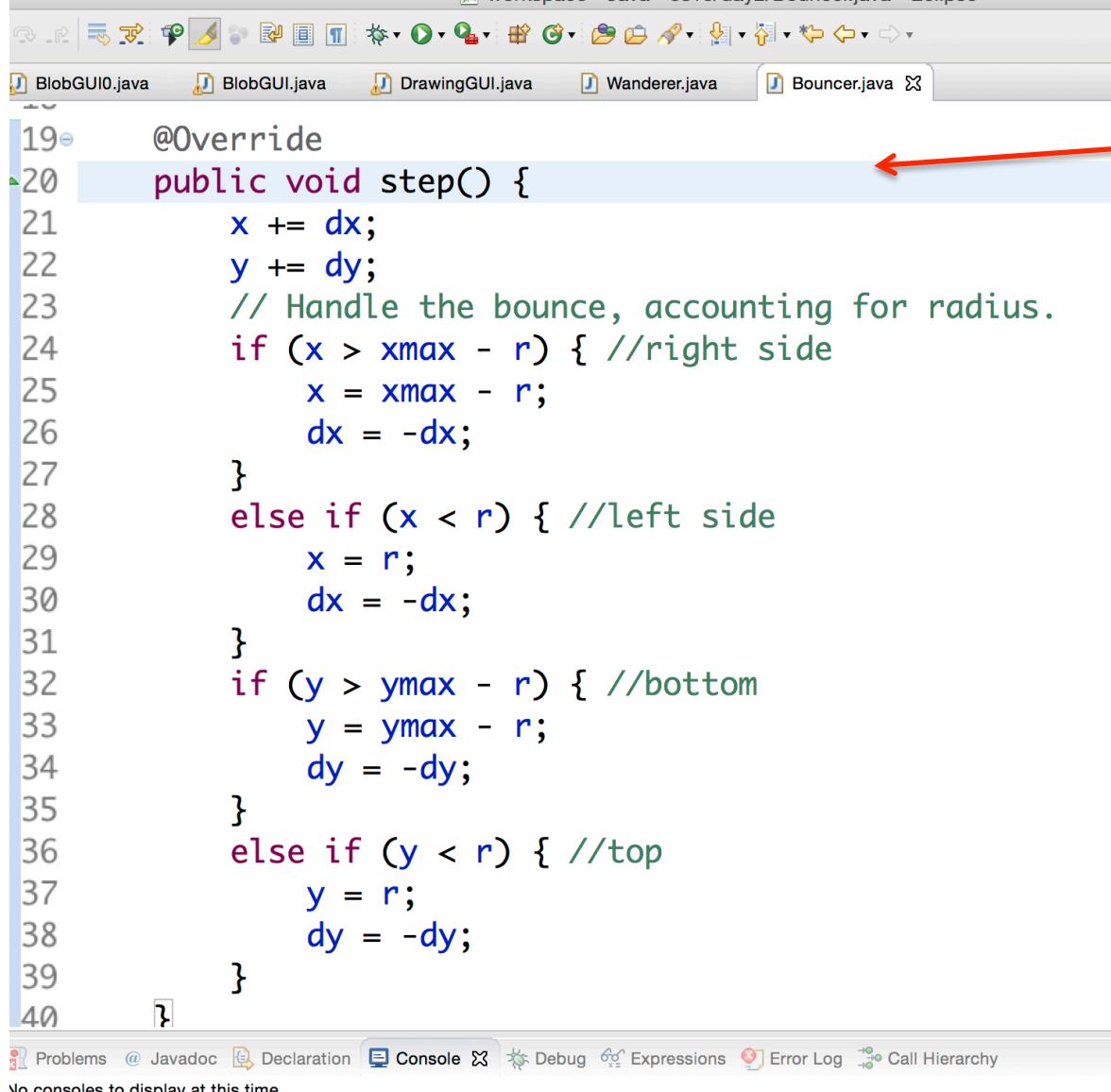
Bouncer extends (inherits from) Blob

Bouncer has new instance variables to define max x and y coordinates

Constructor takes new params for max x and y

When creating a Bouncer, must include four values (not required for other types of Blobs)

A Bouncer “is a” type of Blob that rebounds off sides



```
19  @Override
20  public void step() {
21      x += dx;
22      y += dy;
23      // Handle the bounce, accounting for radius.
24      if (x > xmax - r) { //right side
25          x = xmax - r;
26          dx = -dx;
27      }
28      else if (x < r) { //left side
29          x = r;
30          dx = -dx;
31      }
32      if (y > ymax - r) { //bottom
33          y = ymax - r;
34          dy = -dy;
35      }
36      else if (y < r) { //top
37          y = r;
38          dy = -dy;
39      }
40  }
```

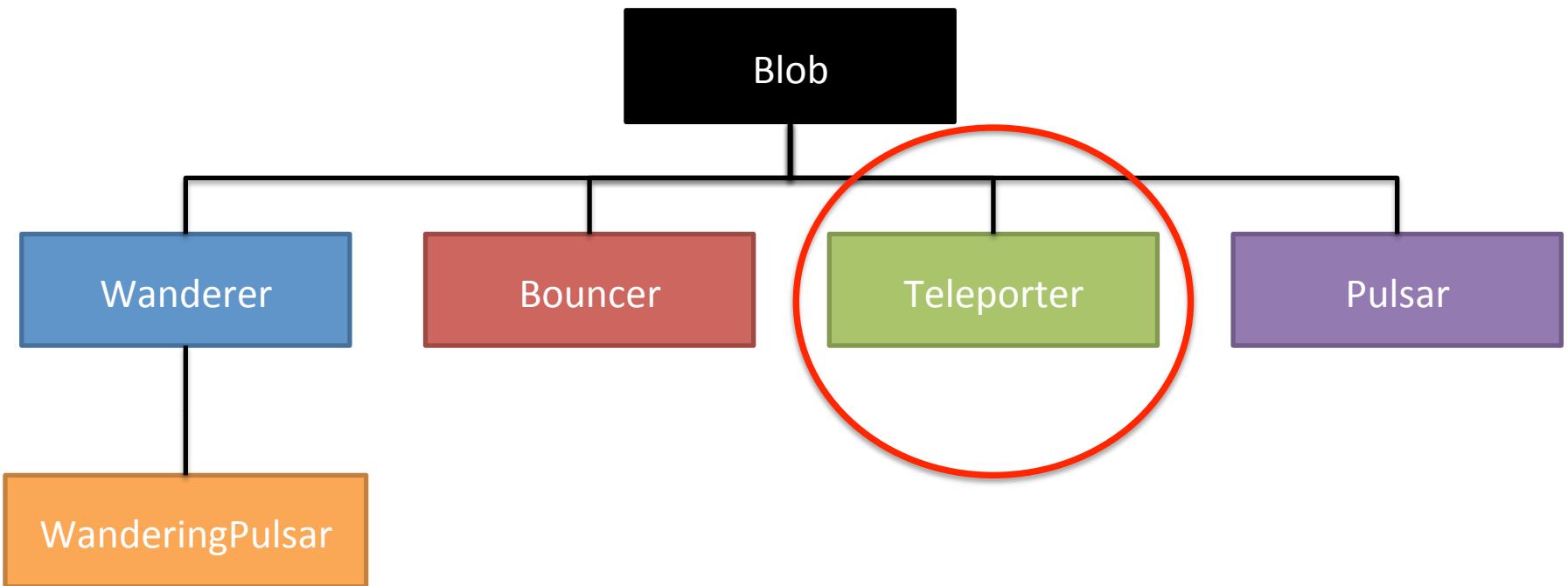
The code shown is the `Bouncer.java` file from the provided screenshot. It contains the implementation of the `step()` method, which moves the blob and handles collisions with the boundaries of its container. A red arrow points to the first line of the `step()` method definition.

Overrides *step()* method

Blob moves in straight line,
but ensures coordinates in
range:

- $r \leq x \leq xmax$
- $r \leq y \leq ymax$
- Reverse sign on dx or dy
if hit wall to rebound in
the opposite direction

A Teleporter can jump to a new location



Teleporter adds a new method other classes do not have

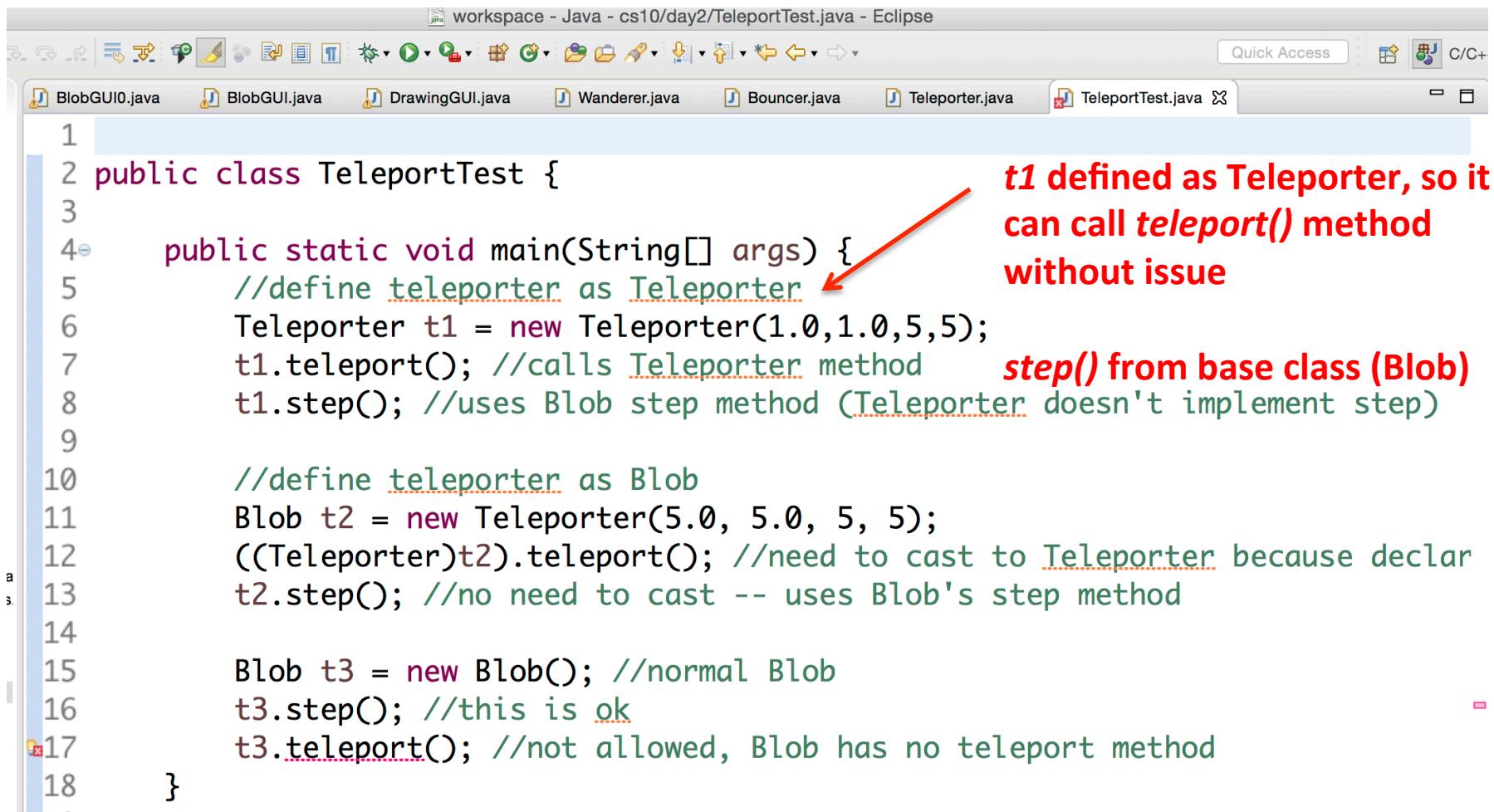
The screenshot shows a Java IDE interface with multiple tabs at the top: BlobGUI0.java, BlobGUI.java, DrawingGUI.java, Wanderer.java, Bouncer.java, Teleporter.java (selected), and TeleportTest.java. The Teleporter.java tab has a red icon.

```
1  /**
2   * A blob with a new method to jump to a random position.
3   */
4  public class Teleporter extends Blob {
5      private int xmax, ymax; // size of teleporting area
6
7  public Teleporter(double x, double y, int xmax, int ymax) {
8      super(x, y);
9      this.xmax = xmax; this.ymax = ymax;
10 }
11
12 /**
13  * Moves the blob to a random new position
14  */
15 public void teleport() {
16     x = Math.random()*xmax;
17     y = Math.random()*ymax;
18 }
19 }
```

Annotations in red:

- An annotation pointing to the `xmax` and `ymax` fields in line 5: **Size of screen stored in xmax and ymax**
- An annotation pointing to the `teleport()` method in line 15: **Only Blobs of type Teleporter can call *teleport()* method (others do not have this method)**
- An annotation pointing to the `x` and `y` assignments in line 16: **What happens when *teleport()* called?
Jump to random x,y location**

Calling teleport() can be tricky



The screenshot shows the Eclipse IDE interface with the title bar "workspace - Java - cs10/day2/TeleportTest.java - Eclipse". The toolbar has various icons for file operations. Below the toolbar, several Java files are listed in the tabs: BlobGUI0.java, BlobGUI.java, DrawingGUI.java, Wanderer.java, Bouncer.java, Teleporter.java, and TeleportTest.java (the active tab). The code editor displays the following Java code:

```
1 public class TeleportTest {  
2  
3     public static void main(String[] args) {  
4         //define teleporter as Teleporter  
5         Teleporter t1 = new Teleporter(1.0,1.0,5,5);  
6         t1.teleport(); //calls Teleporter method  
7         t1.step(); //uses Blob step method (Teleporter doesn't implement step)  
8  
9         //define teleporter as Blob  
10        Blob t2 = new Teleporter(5.0, 5.0, 5, 5);  
11        ((Teleporter)t2).teleport(); //need to cast to Teleporter because declar  
12        t2.step(); //no need to cast -- uses Blob's step method  
13  
14        Blob t3 = new Blob(); //normal Blob  
15        t3.step(); //this is ok  
16        t3.teleport(); //not allowed, Blob has no teleport method  
17    }  
18}
```

A red arrow points from the explanatory text on the right to the line of code where `teleporter` is defined as `Teleporter`.

t1 defined as Teleporter, so it can call `teleport()` method without issue

step() from base class (Blob)

Calling teleport() can be tricky

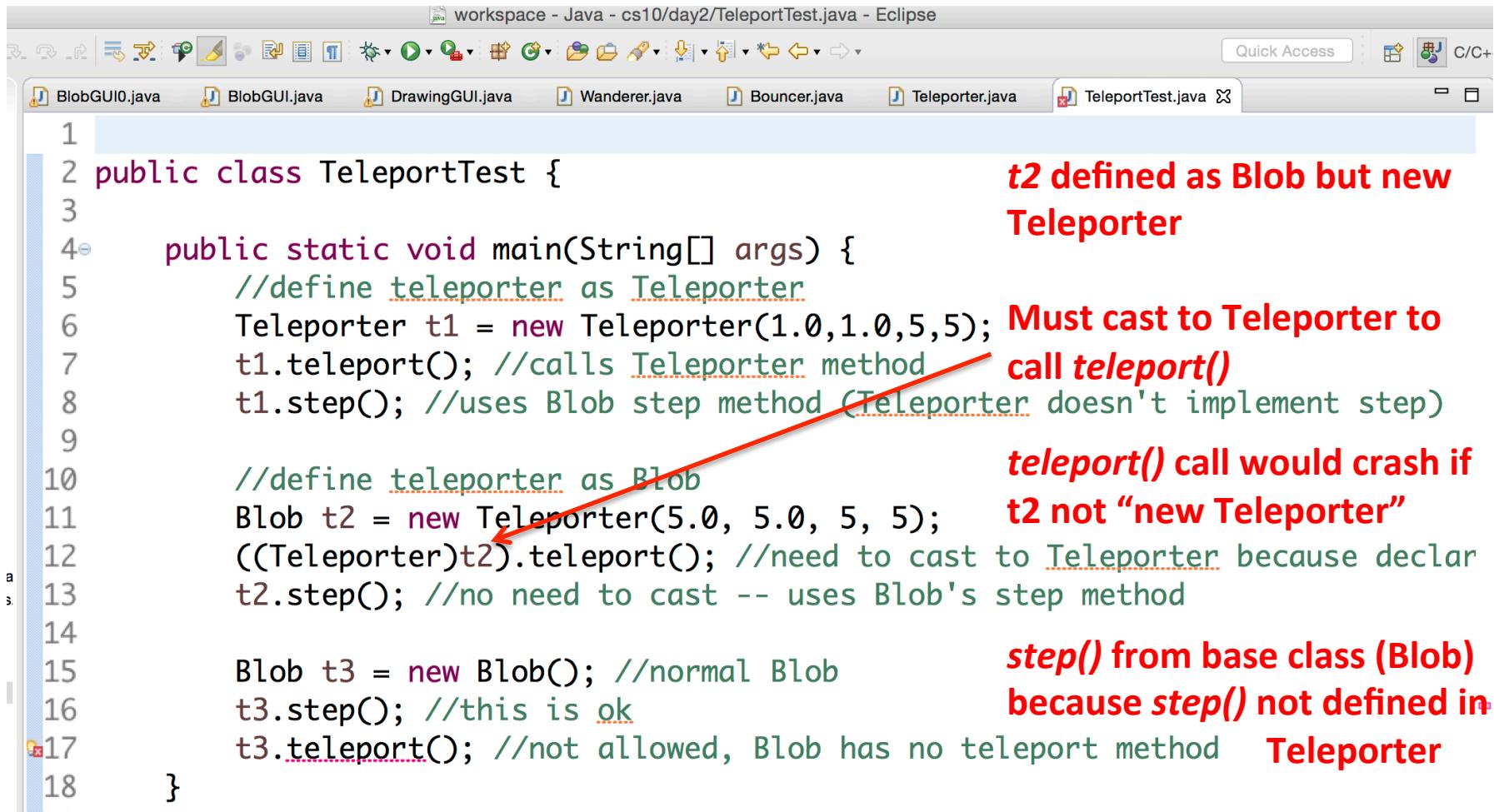
```
workspace - Java - cs10/day2/TeleportTest.java - Eclipse
Quick Access C/C++

1 public class TeleportTest {
2
3     public static void main(String[] args) {
4         //define teleporter as Teleporter
5         Teleporter t1 = new Teleporter(1.0,1.0,5,5);
6         t1.teleport(); //calls Teleporter method
7         t1.step(); //uses Blob step method (Teleporter doesn't implement step)
8
9
10    //define teleporter as Blob
11    Blob t2 = new Teleporter(5.0, 5.0, 5, 5);
12    ((Teleporter)t2).teleport(); //need to cast to Teleporter because declare
13    t2.step(); //no need to cast -- uses Blob's step method
14
15    Blob t3 = new Blob(); //normal Blob
16    t3.step(); //this is ok
17    t3.teleport(); //not allowed, Blob has no teleport method
18 }
```

t2 defined as Blob but new Teleporter

t2 defined as Blob but new Teleporter

Calling teleport() can be tricky



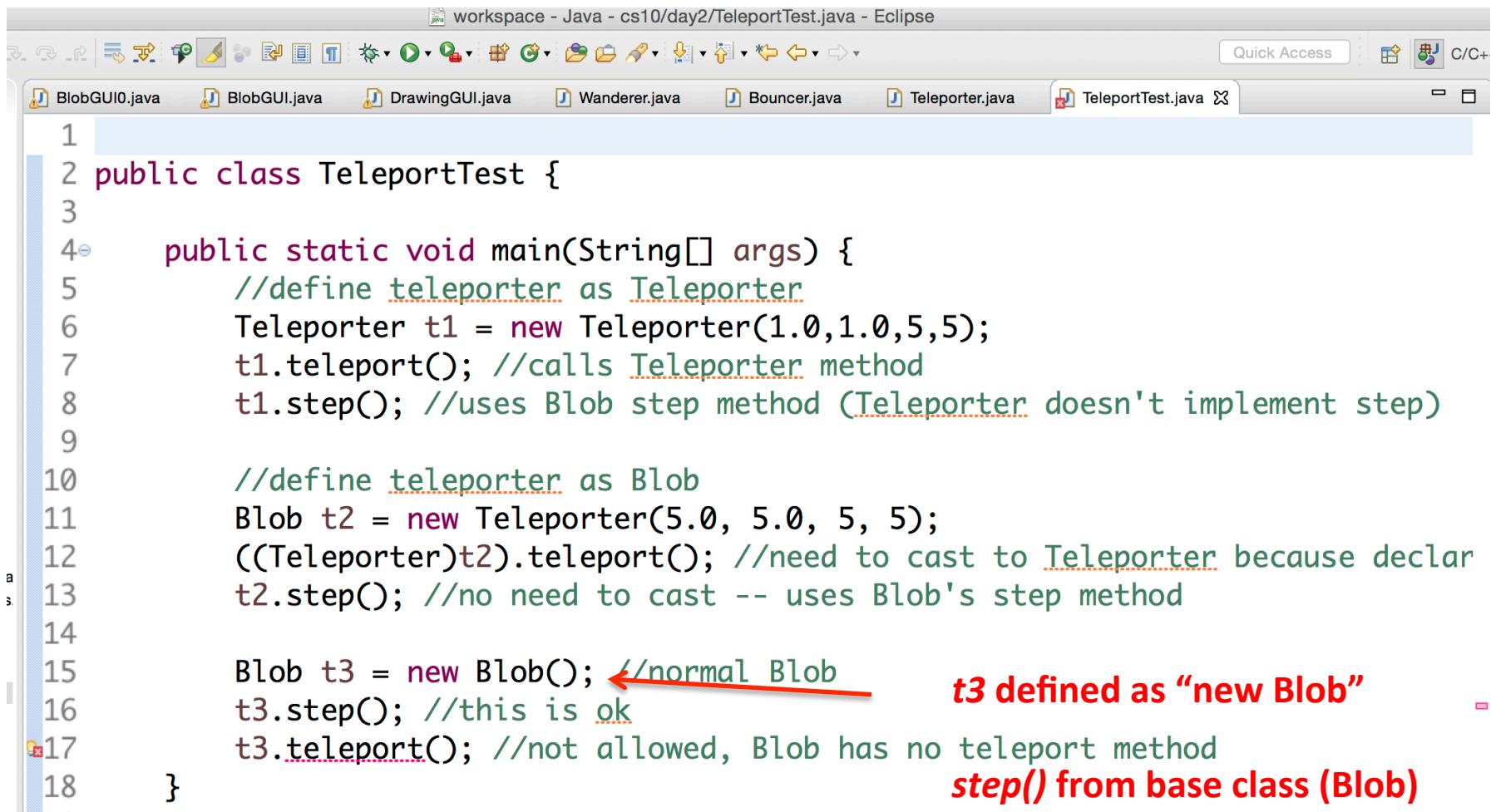
The screenshot shows the Eclipse IDE interface with the title bar "workspace - Java - cs10/day2/TeleportTest.java - Eclipse". The toolbar has various icons for file operations. Below the toolbar is a tab bar with several Java files: BlobGUI0.java, BlobGUI.java, DrawingGUI.java, Wanderer.java, Bouncer.java, Teleporter.java, and TeleportTest.java (the active file). The code editor displays the following Java code:

```
1 public class TeleportTest {  
2  
3     public static void main(String[] args) {  
4         //define teleporter as Teleporter  
5         Teleporter t1 = new Teleporter(1.0,1.0,5,5);  
6         t1.teleport(); //calls Teleporter method  
7         t1.step(); //uses Blob step method (Teleporter doesn't implement step)  
8  
9         //define teleporter as Blob  
10        Blob t2 = new Teleporter(5.0, 5.0, 5, 5);  
11        ((Teleporter)t2).teleport(); //need to cast to Teleporter because declar  
12        t2.step(); //no need to cast -- uses Blob's step method  
13  
14        Blob t3 = new Blob(); //normal Blob  
15        t3.step(); //this is ok  
16        t3.teleport(); //not allowed, Blob has no teleport method  
17    }  
18}
```

Annotations in red text are placed next to specific lines of code:

- "t2 defined as Blob but new Teleporter" points to the line `Blob t2 = new Teleporter(5.0, 5.0, 5, 5);`.
- "Must cast to Teleporter to call teleport()" points to the line `((Teleporter)t2).teleport();`. A red arrow also points from this annotation to the cast operator `((Teleporter)` in the code.
- "teleport() call would crash if t2 not “new Teleporter”" points to the line `((Teleporter)t2).teleport();`.
- "step() from base class (Blob) because step() not defined in Teleporter" points to the line `t3.step();`.

Calling teleport() can be tricky



The screenshot shows the Eclipse IDE interface with the title bar "workspace - Java - cs10/day2/TeleportTest.java - Eclipse". The toolbar has various icons for file operations. Below the toolbar is a tab bar with several Java files: BlobGUI0.java, BlobGUI.java, DrawingGUI.java, Wanderer.java, Bouncer.java, Teleporter.java, and TeleportTest.java (the active file). The code editor displays the following Java code:

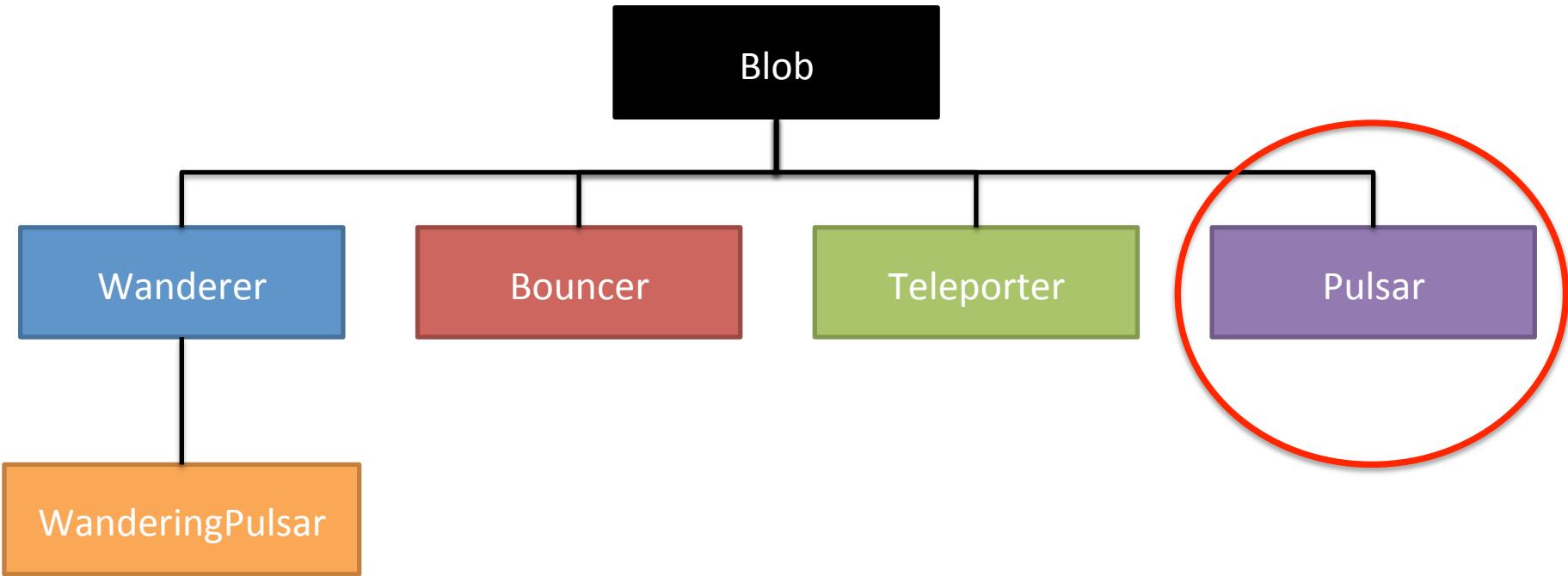
```
1 public class TeleportTest {  
2  
3  
4     public static void main(String[] args) {  
5         //define teleporter as Teleporter  
6         Teleporter t1 = new Teleporter(1.0,1.0,5,5);  
7         t1.teleport(); //calls Teleporter method  
8         t1.step(); //uses Blob step method (Teleporter doesn't implement step)  
9  
10        //define teleporter as Blob  
11        Blob t2 = new Teleporter(5.0, 5.0, 5, 5);  
12        ((Teleporter)t2).teleport(); //need to cast to Teleporter because declar  
13        t2.step(); //no need to cast -- uses Blob's step method  
14  
15        Blob t3 = new Blob(); //normal Blob  
16        t3.step(); //this is ok  
17        t3.teleport(); //not allowed, Blob has no teleport method  
18    }
```

Annotations in red text are present in the code editor:

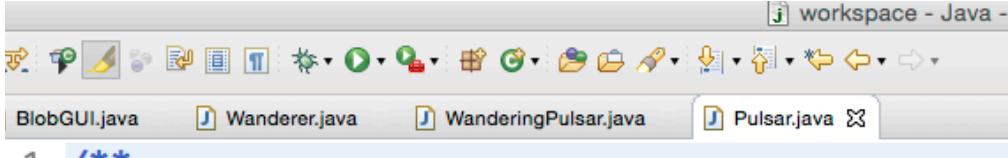
- A red arrow points to the line "t3 = new Blob();" with the text "t3 defined as “new Blob”".
- A red arrow points to the line "t3.teleport();" with the text "step() from base class (Blob)".
- A red text box at the bottom right contains the message "teleport() call not allowed".

teleport() call not allowed

A Pulsar grows and shrinks, but does not move



A Pulsar grows and shrinks, but does not move



```
workspace - Java -  
BlobGUI.java  Wanderer.java  WanderingPulsar.java  Pulsar.java  
1  /*  
2  * A blob that sits still and pulsates (radiates)  
3  */  
4  public class Pulsar extends Blob {  
5      public Pulsar(double x, double y) {  
6          super(x, y);  
7          dr = 1 + Math.random(); // random growth rate  
8      }  
9  
10     @Override  
11     public void step() {  
12         r += dr;  
13         // Make sure radius is within bounds  
14         if (r < 1 || r > 10) {  
15             dr = -dr;  
16             r += dr;  
17         }  
18     }  
19 }  
20
```

Call super constructor

Pick a random radius growth rate between 1 and 2

A Pulsar grows and shrinks, but does not move

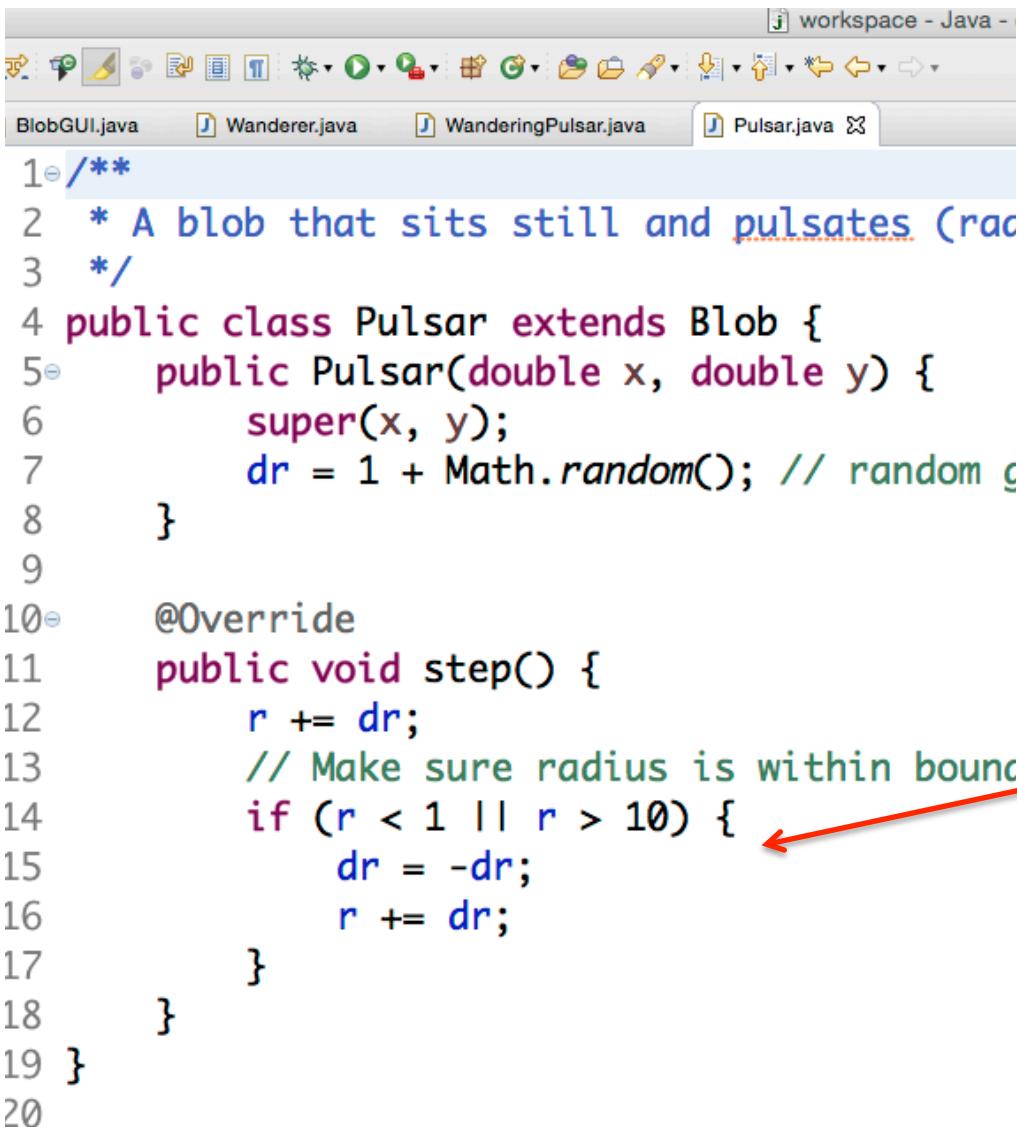
```
workspace - Java -  
BlobGUI.java  Wanderer.java  WanderingPulsar.java  Pulsar.java  
1  /**  
2  * A blob that sits still and pulsates (radiates)  
3  */  
4  public class Pulsar extends Blob {  
5      public Pulsar(double x, double y) {  
6          super(x, y);  
7          dr = 1 + Math.random(); // random growth rate  
8      }  
9  
10     @Override  
11     public void step() {  
12         r += dr; ←  
13         // Make sure radius is within bounds  
14         if (r < 1 || r > 10) {  
15             dr = -dr;  
16             r += dr;  
17         }  
18     }  
19 }  
20
```

Call super constructor

Pick a random radius growth rate between 1 and 2

Update radius on *step()*

A Pulsar grows and shrinks, but does not move



The screenshot shows a Java workspace with several files listed in the toolbar: BlobGUI.java, Wanderer.java, WanderingPulsar.java, and Pulsar.java. The Pulsar.java file is currently open and displayed in the editor.

```
1  /**
2   * A blob that sits still and pulsates (radiates).
3   */
4  public class Pulsar extends Blob {
5      public Pulsar(double x, double y) {
6          super(x, y);
7          dr = 1 + Math.random(); // random growth rate
8      }
9
10     @Override
11     public void step() {
12         r += dr;
13         // Make sure radius is within bounds
14         if (r < 1 || r > 10) {
15             dr = -dr;
16             r += dr;
17         }
18     }
19 }
20 }
```

Call super constructor

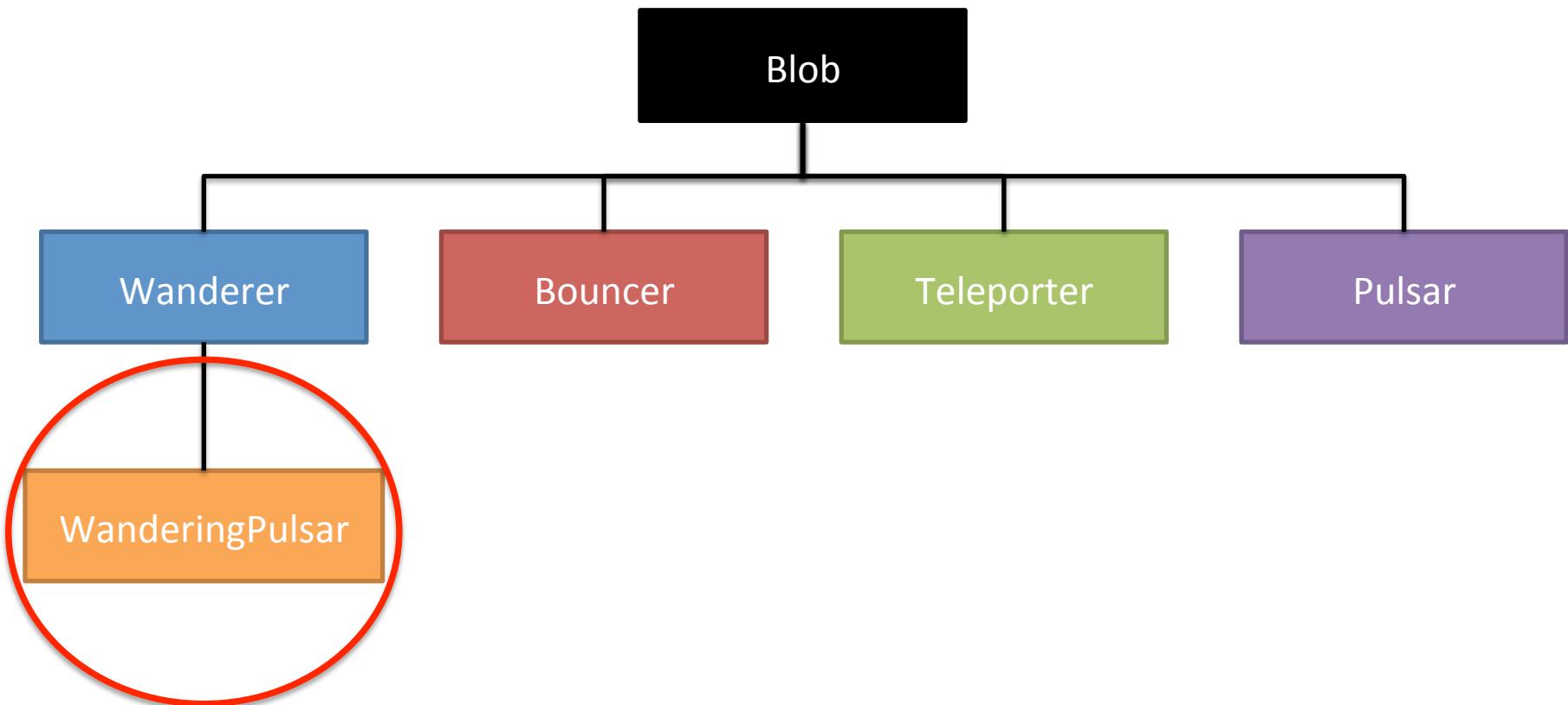
Pick a random radius growth rate between 1 and 2

Update radius on *step()*

Make sure radius doesn't get too big or become negative

If too big or too small, reverse sign on dr to change growing or shrinking rate

A WanderingPulsar inherits from Wanderer (so it moves), and grows and shrinks



A WanderingPulsar inherits from Wanderer (so it moves), and grows and shrinks

```
workspace - Java - cs10/day2/Wandering
BlobGUI.java  Wanderer.java  WanderingPulsar.java  Pulsar.java

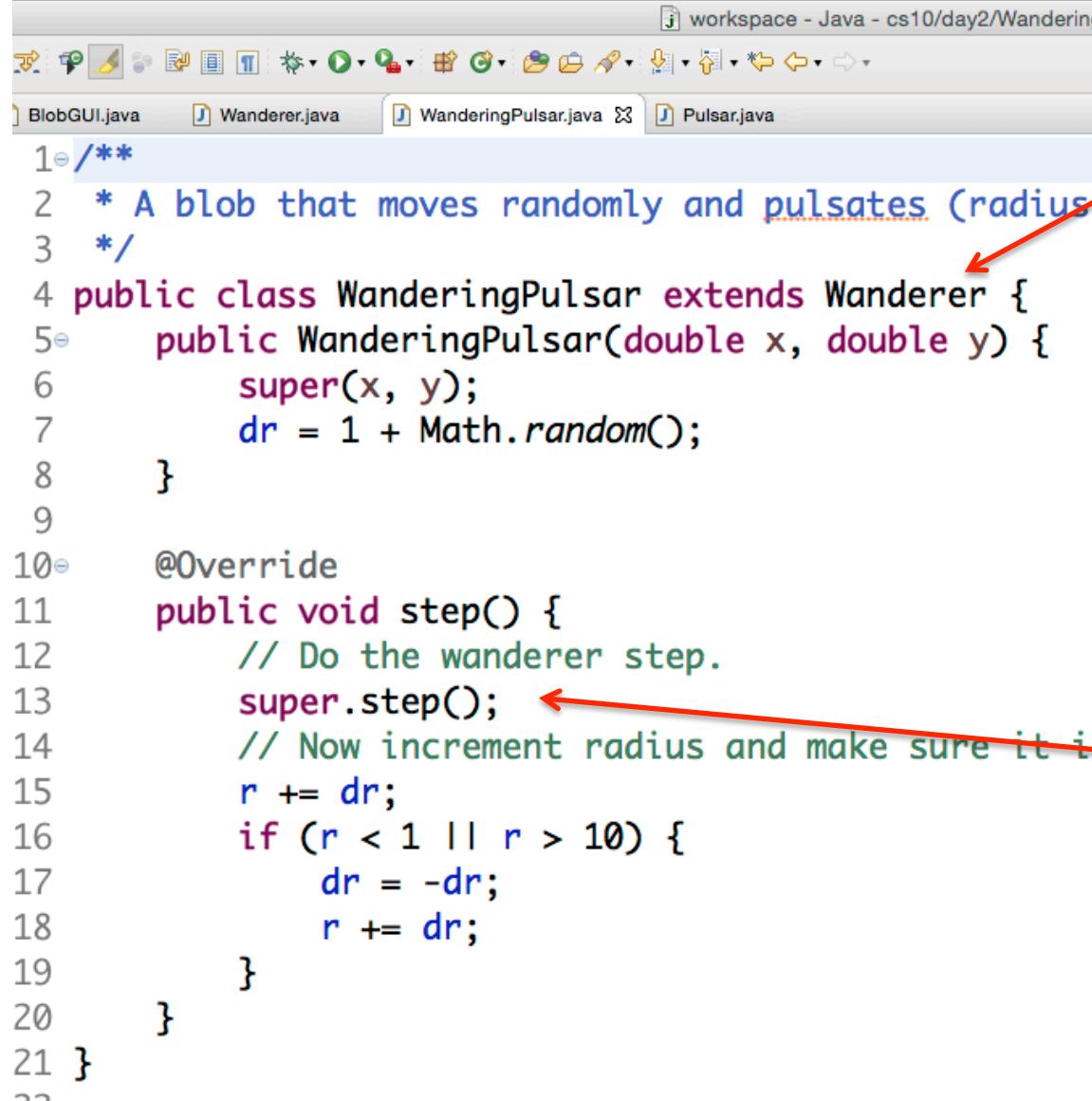
1 /**
2  * A blob that moves randomly and pulsates (radius
3 */
4 public class WanderingPulsar extends Wanderer {
5     public WanderingPulsar(double x, double y) {
6         super(x, y);
7         dr = 1 + Math.random();
8     }
9
10    @Override
11    public void step() {
12        // Do the wanderer step.
13        super.step();
14        // Now increment radius and make sure it is
15        r += dr;
16        if (r < 1 || r > 10) {
17            dr = -dr;
18            r += dr;
19        }
20    }
21 }
```

Inherit from Wanderer

In Java, can only inherit from one super class

We'll need to duplicate a little code to incorporate growing/shrinking

A WanderingPulsar inherits from Wanderer (so it moves), and grows and shrinks



The screenshot shows the Eclipse IDE interface with the title bar "workspace - Java - cs10/day2/Wandering". Below the title bar, there are tabs for "BlobGUI.java", "Wanderer.java", "WanderingPulsar.java" (which is the active tab), and "Pulsar.java". The code editor displays the "WanderingPulsar.java" file. The code defines a class "WanderingPulsar" that extends "Wanderer". It includes a constructor, an overridden "step()" method, and logic for growing and shrinking the radius.

```
1  /**
2  * A blob that moves randomly and pulsates (radius
3  */
4 public class WanderingPulsar extends Wanderer {
5     public WanderingPulsar(double x, double y) {
6         super(x, y);
7         dr = 1 + Math.random();
8     }
9
10    @Override
11    public void step() {
12        // Do the wanderer step.
13        super.step(); ←
14        // Now increment radius and make sure it is
15        r += dr;
16        if (r < 1 || r > 10) {
17            dr = -dr;
18            r += dr;
19        }
20    }
21 }
```

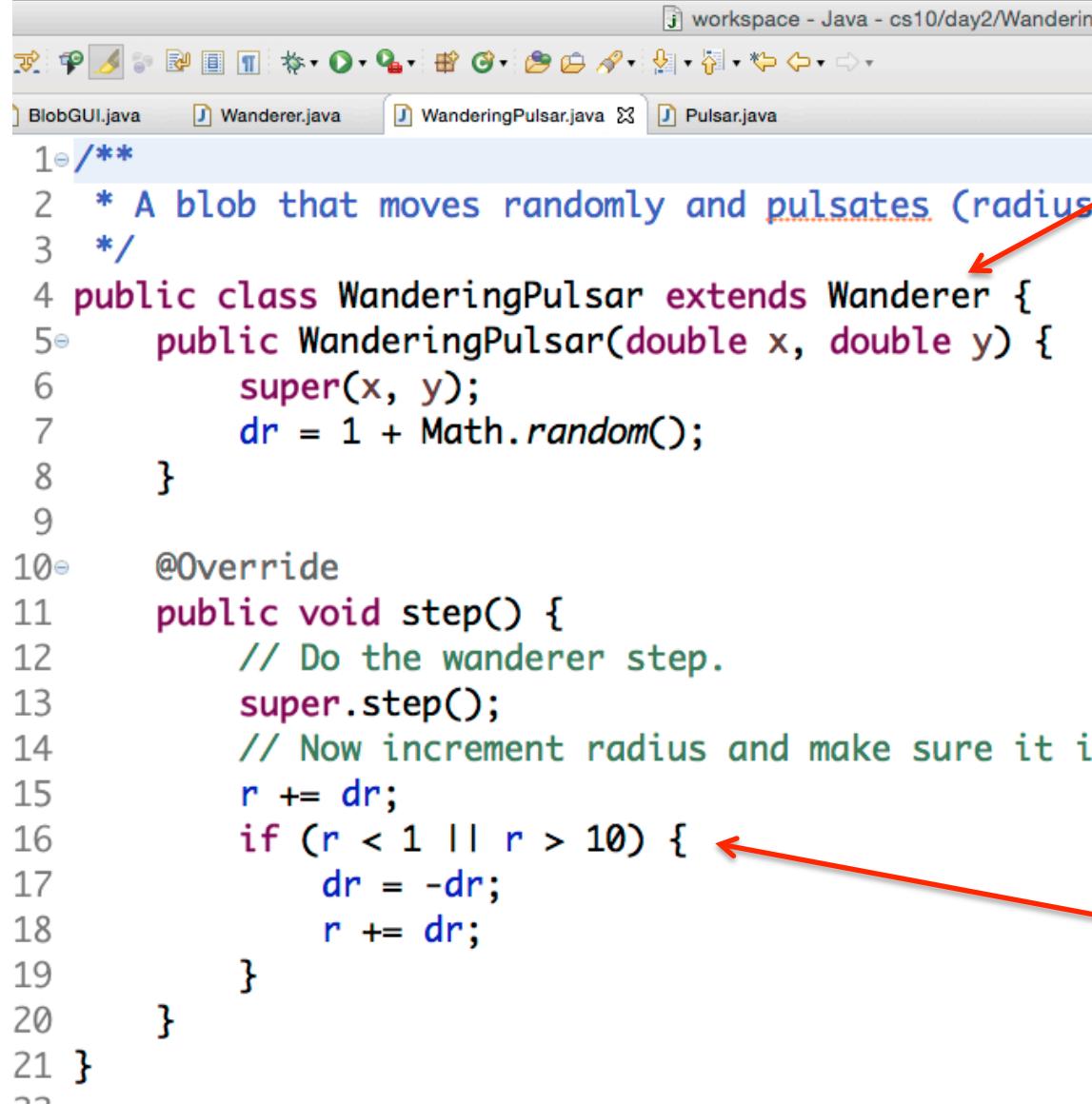
Inherit from Wanderer

In Java, can only inherit from one super class

We'll need to duplicate a little code to incorporate growing/shrinking

Call Wanderer's *step()* to move Blob

A WanderingPulsar inherits from Wanderer (so it moves), and grows and shrinks



The screenshot shows the Eclipse IDE interface with the title bar "workspace - Java - cs10/day2/Wandering". Below the title bar, there are tabs for "BlobGUI.java", "Wanderer.java", "WanderingPulsar.java" (which is the active tab), and "Pulsar.java". The code editor displays the WanderingPulsar.java file with the following content:

```
1  /**
2  * A blob that moves randomly and pulsates (radius
3  */
4  public class WanderingPulsar extends Wanderer {
5      public WanderingPulsar(double x, double y) {
6          super(x, y);
7          dr = 1 + Math.random();
8      }
9
10     @Override
11     public void step() {
12         // Do the wanderer step.
13         super.step();
14         // Now increment radius and make sure it is
15         r += dr;
16         if (r < 1 || r > 10) {
17             dr = -dr;
18             r += dr;
19         }
20     }
21 }
```

Inherit from Wanderer

In Java, can only inherit from one super class

We'll need to duplicate a little code to incorporate growing/shrinking

Call Wanderer's **step()** to move Blob

Update radius, make sure doesn't get too big or become negative (had to duplicate code because couldn't also inherit from Pulsar)

Agenda

1. Objects vs. primitives
2. Inheritance
3. Quick review of Blobs
4. Using Inheritance to create different types of Blobs
5. Graphical User Interface (GUI) programming

Demo: BlobGUI.java

BlobGUI.java

- Press key to change Blob type
 - B for Bouncer
 - P for Pulsar
 - T for Teleporter
 - W for Wanderer
 - U for WanderingPulsar
- Click mouse to place Blob on screen
- Blob moves each time a timer ticks (every 100ms)

GUI programs do not run straight through, instead they respond to *events*

GUI programs

- Respond to user *events* such as mouse or key presses, rather than running straight through to completion
- Can also respond to non-user initiated events such as timer ticking
- When event occurs, it triggers code responsible for *handling* event, then goes back to waiting for next event
- Java has a lot of functionality, I've provided a simple “wrapper” in *DrawingGUI* to set up GUI display and provide functionality
- Extend *DrawingGUI* in your driver classes to inherit that functionality
- Feel free to look around *DrawingGUI*, but don't feel like you need to “own it”!

DrawingGUI tries to handle a lot of GUI set up messiness for us

Conceptual



- Java provides GUI code
- Somewhat complicated
- Learning the specifics of Java's GUI "machinery" not really the point of this course
- Wrapper that inherits from Java's *JFrame* machinery
- Sets up GUI and provides functionality we can use:
 - *handleMousePress()*
 - *handleKeyPress()*
 - *handleTimer()*
 - *repaint()*
 - *draw()*
- Inherit from *DrawingGUI*
- Override
 - *handleMousePress()*
 - *handleKeyPress()*
 - *handleTimer()*
 - *draw()*
- Get call backs when events (e.g., mouse press) happen
- Can trigger GUI *repaint()*
- *repaint()* calls *draw()*

DrawingGUI.java has a number of useful methods, these are the main ones

Useful DrawingGUI methods

Method	Description
handleMousePress (x, y)	<ul style="list-style-type: none">• Responds to mouse press• Gets x, y coordinates where mouse was located when pressed

DrawingGUI.java has a number of useful methods, these are the main ones

Useful DrawingGUI methods

Method	Description
handleMousePress (x, y)	<ul style="list-style-type: none">• Responds to mouse press• Gets x, y coordinates where mouse was located when pressed
handleKeyPress (key)	<ul style="list-style-type: none">• Responds to keyboard key presses• Gets the character pressed

DrawingGUI.java has a number of useful methods, these are the main ones

Useful DrawingGUI methods

Method	Description
handleMousePress (x, y)	<ul style="list-style-type: none">• Responds to mouse press• Gets x, y coordinates where mouse was located when pressed
handleKeyPress (key)	<ul style="list-style-type: none">• Responds to keyboard key presses• Gets the character pressed
handleTimer ()	<ul style="list-style-type: none">• Callback when timer ticks• Ask the Blob to <code>step ()</code>, then <code>repaint ()</code>

DrawingGUI.java has a number of useful methods, these are the main ones

Useful DrawingGUI methods

Method	Description
handleMousePress (x, y)	<ul style="list-style-type: none">• Responds to mouse press• Gets x, y coordinates where mouse was located when pressed
handleKeyPress (key)	<ul style="list-style-type: none">• Responds to keyboard key presses• Gets the character pressed
handleTimer ()	<ul style="list-style-type: none">• Callback when timer ticks• Ask the Blob to <code>step()</code>, then <code>repaint()</code>
repaint ()	<ul style="list-style-type: none">• Refreshes the GUI display• We call it when the underlying state changes (e.g., Blob moves)• OS may also call it to manage windows• <code>repaint()</code> calls our <code>draw()</code> function each time it is called

DrawingGUI.java has a number of useful methods, these are the main ones

Useful DrawingGUI methods

Method	Description
handleMousePress (x, y)	<ul style="list-style-type: none">• Responds to mouse press• Gets x, y coordinates where mouse was located when pressed
handleKeyPress (key)	<ul style="list-style-type: none">• Responds to keyboard key presses• Gets the character pressed
handleTimer ()	<ul style="list-style-type: none">• Callback when timer ticks• Ask the Blob to <code>step()</code>, then <code>repaint()</code>
repaint ()	<ul style="list-style-type: none">• Refreshes the GUI display• We call it when the underlying state changes (e.g., Blob moves)• OS may also call it to manage windows• <code>repaint()</code> calls our <code>draw()</code> function each time it is called
draw(Graphics)	<ul style="list-style-type: none">• This is how we draw our portion of the screen (e.g., the Blob)• Inherit from <code>DrawingGUI</code> base class• Should call the Blob's <code>draw()</code> function to display the Blob (Blob's <code>draw()</code> calls <code>fillOval()</code> to display the Blob)

BlobGUI0.java: DrawingGUI handles most of the hard parts, inherit from it and use it

workspace - Java - cs10/day2/BlobGUI0.java - Eclipse

Quick Access

BlobGUI class inherits from DrawingGUI to get access to methods

```
10 public class BlobGUI0 extends DrawingGUI {  
11     private static final int width=800, height=600; // size of the world  
12  
13     private Blob blob; // Declares a Blob instance variable  
14  
15     public BlobGUI0() { // Use DrawingGUI to set up graphical display (so we don't have to)  
16         super("Animated Blob", width, height); // Creates new Blob (declared above)  
17  
18         // Create blob at the center.  
19         blob = new Blob(width/2, height/2); // Puts Blob in center or display  
20         blob.setVelocity(0.5, 0.5); // Tells Blob to move in small steps // What happens if this isn't  
21  
22         // Timer drives the animation.  
23         startTimer(); //ticks every 100ms by default  
24     } // Starts timer that ticks every 100ms  
25  
26     /** Found in DrawingGUI  
27      * DrawingGUI method, here just drawing the blob  
28  */
```

BlobGUI0.java: DrawingGUI handles most of the hard parts, inherit from it and use it

```
 64  public static void main(String[] args) {  
 65      SwingUtilities.invokeLater(new Runnable() {  
 66          public void run() {  
 67              new BlobGUI0();  
 68          }  
 69      });  
 70  }  
 71 }
```

- 
- *main()* is kind of ugly, but it's common “boilerplate”
 - Calls our constructor (from last slide)
 - Starts the GUI running
 - Waits for events such as mouse/key presses or timer ticks

BlobGUI0.java: DrawingGUI handles most of the hard parts, inherit from it and use it

The screenshot shows the Eclipse IDE interface with the title bar "workspace - Java - cs10/day2/BlobGUI0.java - Eclipse". The toolbar has various icons for file operations, code navigation, and toolbars. Below the toolbar, two tabs are visible: "BlobGUI0.java" and "BlobGUI.java". The code editor displays the following Java code:

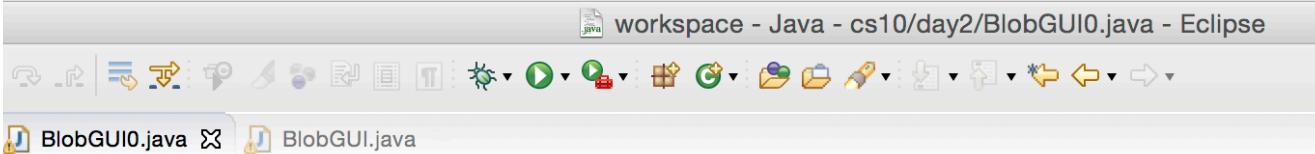
```
28     */
29     @Override
30     public void draw(Graphics g) {
31         // Ask the blob to draw itself.
32         blob.draw(g);
33     }
34
35 /**
36 * DrawingGUI method, here having the blob take a step
37 */
38 @Override
39 public void handleTimer() {
40     // Ask the blob to move itself.
41     blob.step();
42     // Now update the GUI.
43     repaint(); //canvas set up in DrawGUI.java initWindow
44 }
45 /**
46 */


```

Annotations in red text and arrows explain specific parts of the code:

- An arrow points from the text "draw() asks the Blob to draw itself (see Blob.java) on the Graphics screen g" to the line `blob.draw(g);`.
- An arrow points from the text "handleTimer() called whenever timer ticks" to the line `handleTimer()`.
- An arrow points from the text "Called a “callback” function, because method is called back whenever a timer event happens" to the line `handleTimer()`.
- A bulleted list below the annotations provides more details:
 - *step()* tells Blob to move
 - *repaint()* (from DrawingGUI) calls our *draw()* method

BlobGUI0.java: DrawingGUI handles most of the hard parts, inherit from it and use it

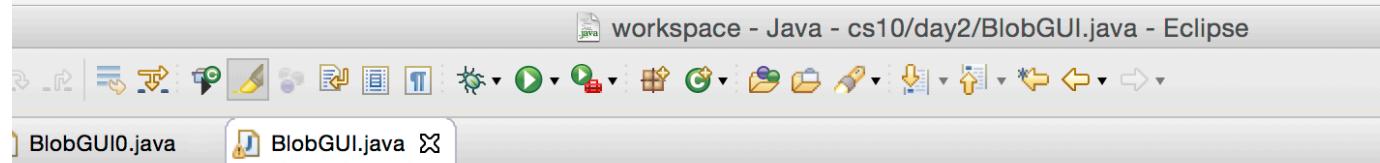


The screenshot shows the Eclipse IDE interface with the title bar "workspace - Java - cs10/day2/BlobGUI0.java - Eclipse". Below the title bar is the toolbar with various icons. The editor area displays the code for BlobGUI0.java. The code defines a method handleMousePress that checks if a mouse click is inside a blob and prints "back off!" if true, or moves the blob if false. It also calls repaint(). A red arrow points from the text "back off!" in the code to a bulleted list of annotations.

```
45  
46*     /**  
47*      * DrawingGUI method, here either detecting if  
48*      */  
49@override  
50 public void handleMousePress(int x, int y) {  
51     if (blob.contains(x, y)) {  
52         // Clicked on it  
53         System.out.println("back off!");  
54     }  
55     else {  
56         // Moved it  
57         blob.setX(x);  
58         blob.setY(y);  
59         // Redraw with moved blob  
60         repaint();  
61     }  
62 }
```

- **handleMousePress()** called whenever mouse is pressed
- Checks to see if x,y coordinates where mouse pressed is inside blob
 - If true, print “back off!”
 - Else move Blob to location of mouse press
- Call *repaint()* which will in turn call our *draw()*

BlobGUI.java: Adds more functionality, monitors for key press, changes Blob type

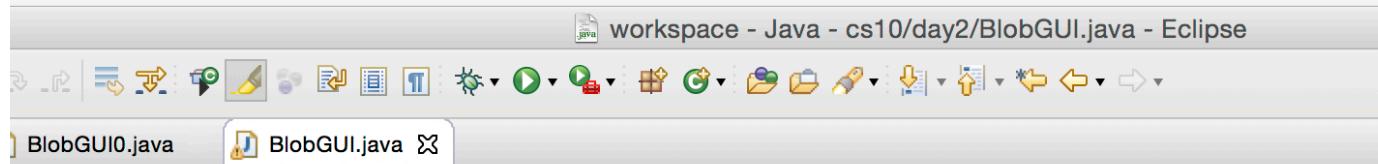


```
workspace - Java - cs10/day2/BlobGUI.java - Eclipse
File Edit View Insert Run Project Properties Help
BlobGUI0.java BlobGUI.java

66  @Override
67  public void handleKeyPress(char k) {
68      System.out.println("Handling key '" + k + "'");
69      if (k == 'f') { // faster
70          if (delay > 1) delay /= 2;
71          setTimerDelay(delay);
72          System.out.println("delay:" + delay);
73      }
74      else if (k == 's') { // slower
75          delay *= 2;
76          setTimerDelay(delay);
77          System.out.println("delay:" + delay);
78      }
79      else { // blob type
80          System.out.println("blob type:" + k);
81          blobType = k;
82      }
83  }
```

- **handleKeyPress()** called back whenever key is pressed

BlobGUI.java: Adds more functionality, monitors for key press, changes Blob type

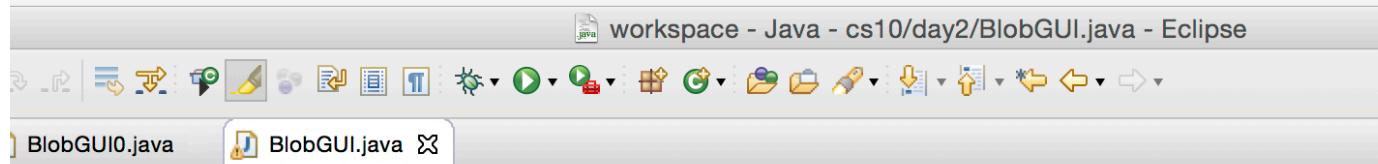


```
workspace - Java - cs10/day2/BlobGUI.java - Eclipse
File Edit View Insert Run Project Properties Help
BlobGUI0.java BlobGUI.java

66  @Override
67  public void handleKeyPress(char k) {
68      System.out.println("Handling key '" + k + "'");
69      if (k == 'f') { // faster
70          if (delay > 1) delay /= 2;
71          setTimerDelay(delay);
72          System.out.println("delay:" + delay);
73      }
74      else if (k == 's') { // slower
75          delay *= 2;
76          setTimerDelay(delay);
77          System.out.println("delay:" + delay);
78      }
79      else { // blob type
80          System.out.println("blob type:" + k);
81          blobType = k;
82      }
83  }
```

- *handleKeyPress()* called back whenever key is pressed
- Can speed up/slow down timer

BlobGUI.java: Adds more functionality, monitors for key press, changes Blob type



```
workspace - Java - cs10/day2/BlobGUI.java - Eclipse
BlobGUI0.java BlobGUI.java

66  @Override
67  public void handleKeyPress(char k) {
68      System.out.println("Handling key '" + k + "'");
69      if (k == 'f') { // faster
70          if (delay > 1) delay /= 2;
71          setTimerDelay(delay);
72          System.out.println("delay:" + delay);
73      }
74      else if (k == 's') { // slower
75          delay *= 2;
76          setTimerDelay(delay);
77          System.out.println("delay:" + delay);
78      }
79      else { // blob type
80          System.out.println("blob type:" + k);
81          blobType = k;
82      }
83  }
```

- *handleKeyPress()* called back whenever key is pressed
- Can speed up/slow down timer
- Can also change Blob type
- Next time mouse clicked a blob of type *k* is shown

BlobGUI.java: Adds more functionality, monitors for key press, changes Blob type

```
31@ 31 @Override  
32 32 public void handleMousePress(int x, int y) {  
33 33     if (blob.contains(x, y)) {  
34 34         System.out.println("back off!");  
35 35         return;  
36 36     }  
37 37     if (blobType=='0') {  
38 38         blob = new Blob(x,y);  
39 39     }  
40 40     else if (blobType=='b') {  
41 41         blob = new Bouncer(x,y,width,height);  
42 42     }  
43 43     else if (blobType=='p') {  
44 44         blob = new Pulsar(x,y);  
45 45     }  
46 46     else if (blobType=='t') {  
47 47         blob = new Teleporter(x,y,width,height);  
48 48     }  
49 49     else if (blobType=='w') {  
50 50         blob = new Wanderer(x,y);  
51 51     }  
52 52     else if (blobType=='u') {  
53 53         blob = new WanderingPulsar(x,y);  
54 54     }  
55 55     else {  
56 56         System.err.println("Unknown blob type "+blobType);  
57 57     }  
58 58  
59 59     // Redraw with new blob  
60 60     repaint();  
61 61 }
```

When mouse clicked,
set Blob to new Blob
type based on last key
press, then repaint

Old Blob is garbage
collected (memory
returned to OS)

