CSCI 132:
Basic Data Structures and Algorithms

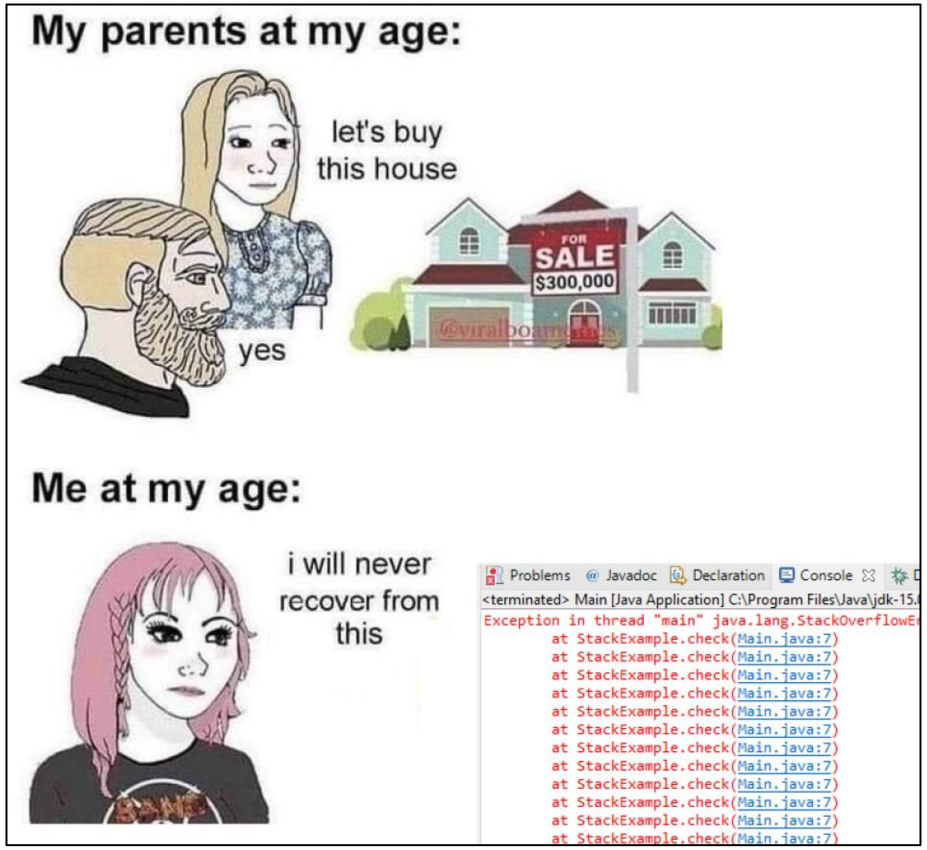
References, Debugging, Program 1

Reese Pearsall & Iliana Castillon

Fall 2024

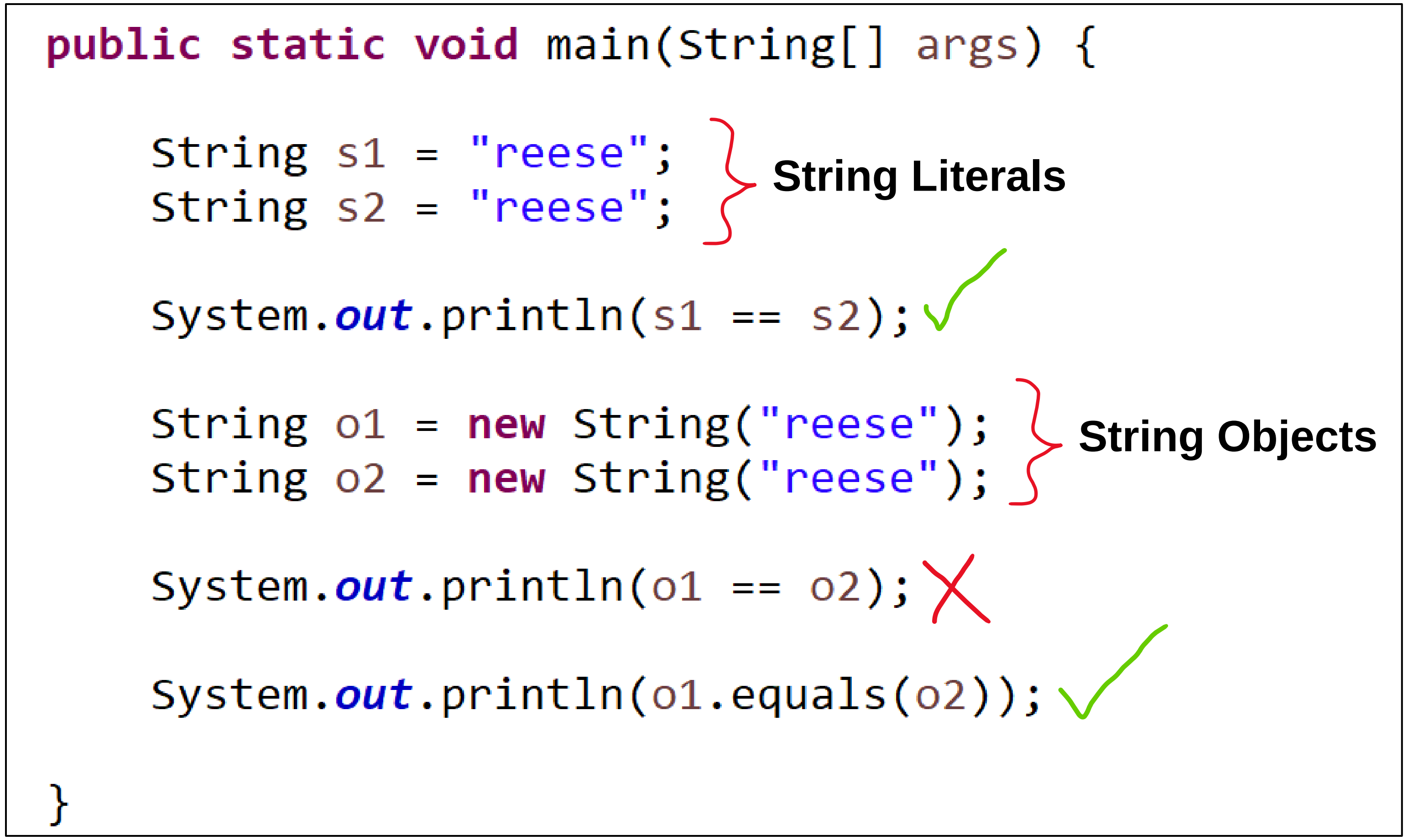
Announcements

- Program 1 posted, due Friday 9/20 @ 11:59 PM
- Lab 3 will be posted shortly after class today



```
<terminated> Main [Java Application] C:\Program Files\Java\jdk-15.
Exception in thread "main" java.lang.StackOverflowE
        at StackExample.check(Main.java:7)
        at StackExample.check(Main.java:7)
        at StackExample.check(Main.java:7)
        at StackExample.check(Main.java:7)
        at StackExample.check(Main.java:7)
        at StackExample.check(Main.java:7)
        at StackExample.check(Main.java:7)
        at StackExample.check(Main.java:7)
        at StackExample.check(Main.java:7)
        at StackExample.check(Main.java:7)
        at StackExample.check(Main.java:7)
        at StackExample.check(Main.java:7)
```

Program 1



```java
public static void main(String[] args) {

    String s1 = "reese";
    String s2 = "reese";

    System.out.println(s1 == s2);

    String o1 = new String("reese");
    String o2 = new String("reese");

    System.out.println(o1 == o2);

    System.out.println(o1.equals(o2));

}
```

When comparing two objects, the == operator will check if the two reference values are pointing to the same object

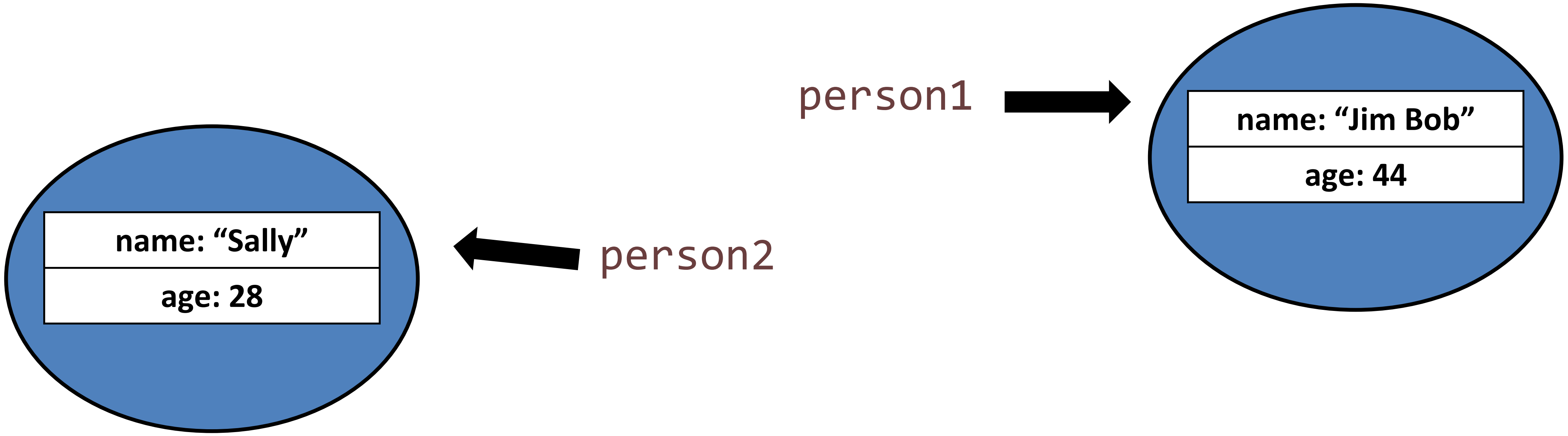
When using **string literals**, Java won't create two separate objects for each string, so sometimes == will work

If we make them **String objects**, now == will not work because these are two different String objects

When comparing Strings, you should still always use **.equals()**

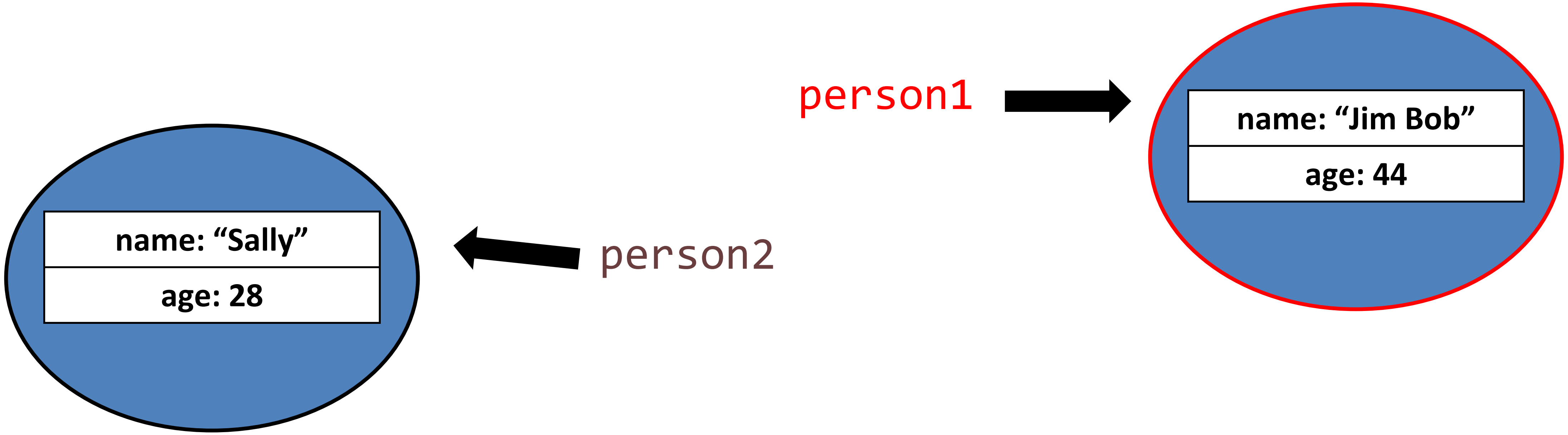
```
public class ReferencesDemo {
        public static void main(String[] args) {

                Person person1 = new Person("Jim Bob", 44);
                Person person2 = new Person("Sally", 28);
        }
}
```

`person1` and `person2` are **references** to a Person object



```
public class ReferencesDemo {
        public static void main(String[] args) {

                Person person1 = new Person("Jim Bob", 44);
                Person person2 = new Person("Sally", 28);
                person1.changeName("Jack");

        }
}
```

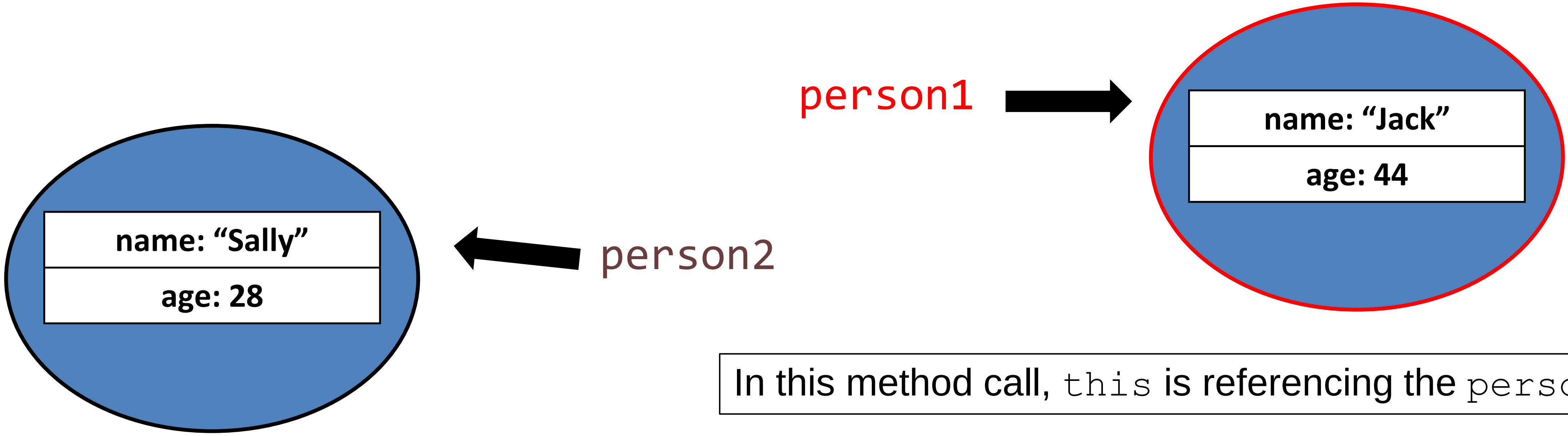
`person1` and `person2` are **references** to a Person object



```
public class ReferencesDemo {
        public static void main(String[] args) {

                Person person1 = new Person("Jim Bob", 44);
                Person person2 = new Person("Sally", 28);
                person1.changeName("Jack");

        }
}
```

```
public void changeName(String newName) {
    this.name = newName;
}
```

`person1` and `person2` are **<u>references</u>** to a Person object

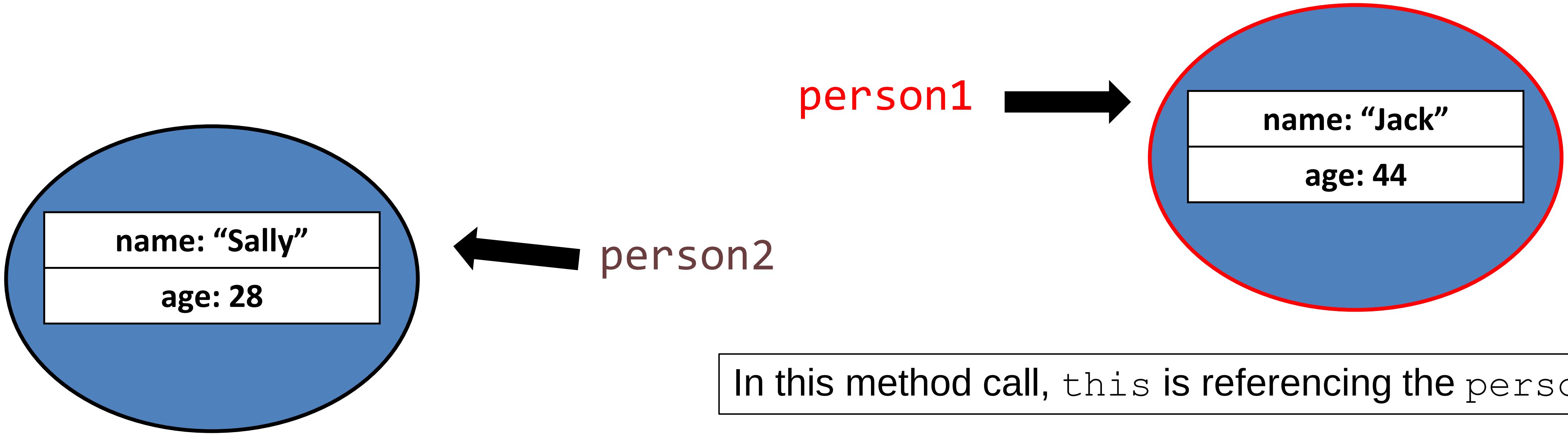


In this method call, `this` is referencing the `person1` object

```java
public class ReferencesDemo {
       public static void main(String[] args) {

              Person person1 = new Person("Jim Bob", 44);
              Person person2 = new Person("Sally", 28);

              Person person3 = person1;
       }
}
```

Suppose we create a new reference variable and link it to an existing object

```java
public void changeName(String newName) {
    this.name = newName;
}
```



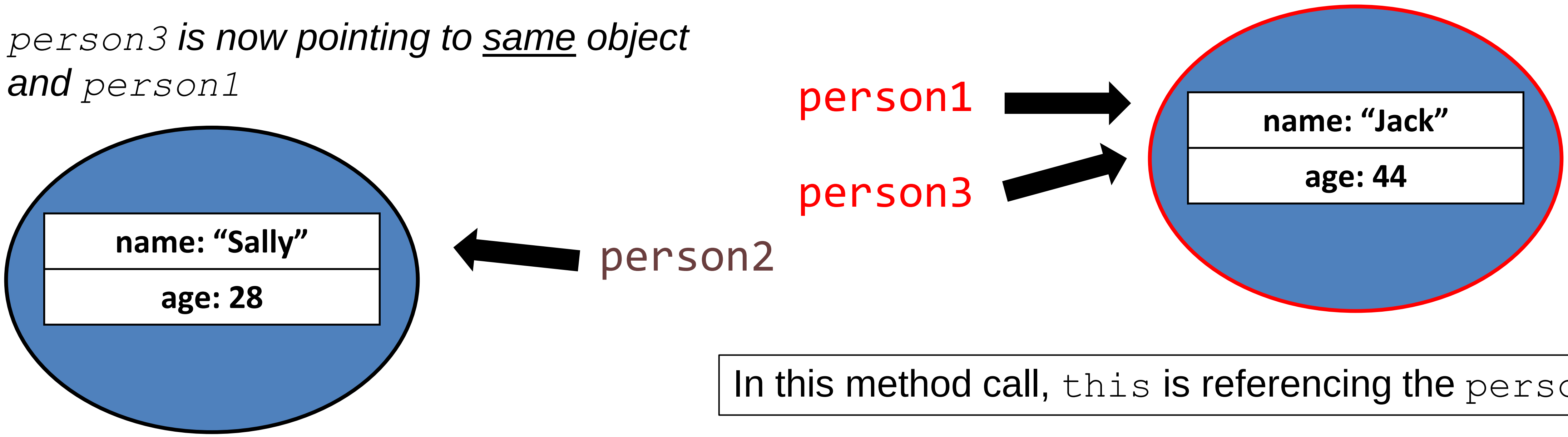
In this method call, `this` is referencing the `person1` object

```
public class ReferencesDemo {
        public static void main(String[] args) {

                Person person1 = new Person("Jim Bob", 44);
                Person person2 = new Person("Sally", 28);

                Person person3 = person1;
        }
}
```

```
public void changeName(String newName) {
    this.name = newName;
}
```

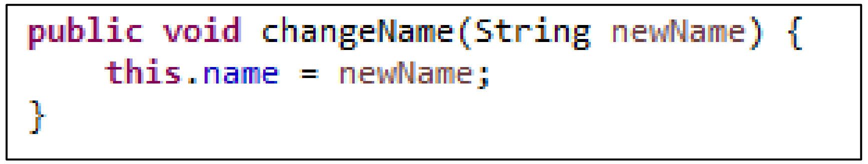
Suppose we create a new reference variable and link it to an existing object

`person3` is now pointing to <u>same</u> object and `person1`



In this method call, `this` is referencing the `person1` object

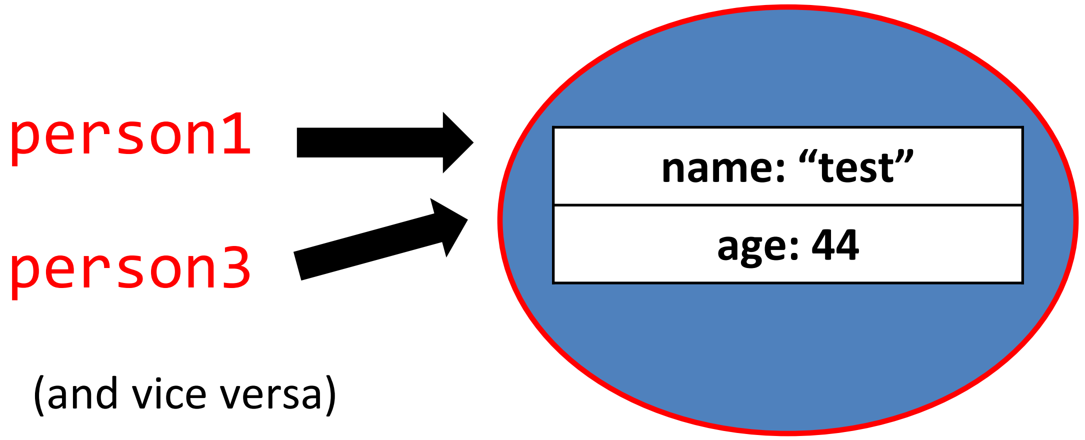
```
public class ReferencesDemo {
	public static void main(String[] args) {

		Person person1 = new Person("Jim Bob", 44);
		Person person2 = new Person("Sally", 28);

		Person person3 = person1;
		person1.changeName("test");
	}
}
```



```
public void changeName(String newName) {
    this.name = newName;
}
```

Suppose we create a new reference variable and link it to an existing object

`person3` is now pointing to same object and `person1`

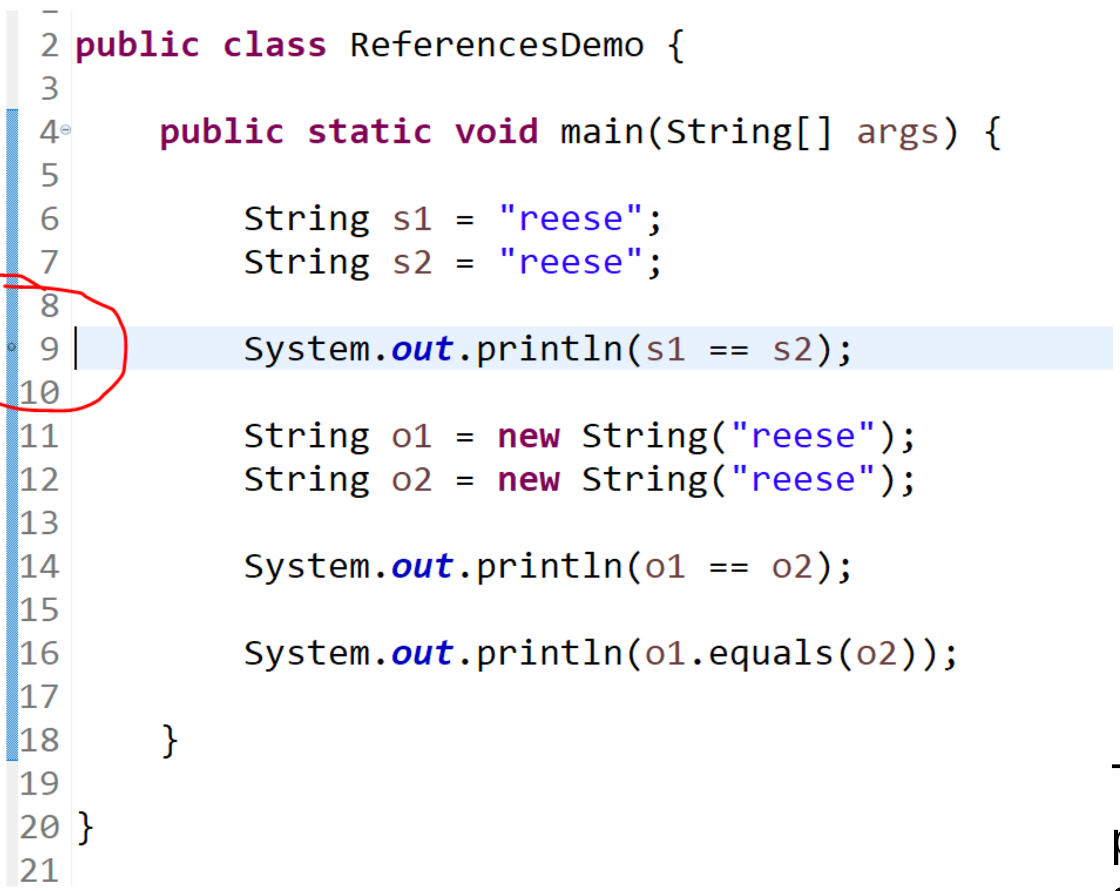


Any changes to `person1` will also update `person3` (and vice versa)

`System.out.println(person1.getName())` → "test"
`System.out.println(person3.getName())` → "test"

Debugging Code

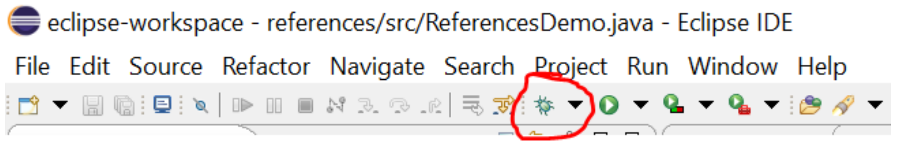
Our IDE has a super nifty **debugger**, which allows us to pause our code, and then step through each line in the control flow.



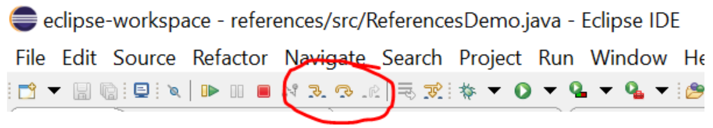
```
public class ReferencesDemo {

    public static void main(String[] args) {

        String s1 = "reese";
        String s2 = "reese";

        System.out.println(s1 == s2);

        String o1 = new String("reese");
        String o2 = new String("reese");

        System.out.println(o1 == o2);

        System.out.println(o1.equals(o2));

    }

}
```

The first thing to do is to place a **breakpoint,** which is where execution will pause at, and debugging will begin

- Usually you try to place the breakpoint where you think things are going wrong


eclipse-workspace - references/src/ReferencesDemo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

Then, press the little green bug icon next to the play button, which will run the debugger and stop at your breakpoint


eclipse-workspace - references/src/ReferencesDemo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window He

Use the “step into” and “step over” buttons to start walking through your code

Debugging Code

Our IDE has a super slick debugger built in to it. I highly recommend learning how to use the debugger tool (see lecture)

Rubber Duck Debugging

Many programmers have had the experience of explaining a problem to someone else, possibly even to someone who knows nothing about programming, and then hitting upon the solution in the process of explaining the problem. In describing what the code is supposed to do and observing what it actually does, any incongruity between these two becomes apparent.[2] More generally, teaching a subject forces its evaluation from different perspectives and can provide a deeper understanding.[3] By using an inanimate object, the programmer can try to accomplish this without having to interrupt anyone else, and with better results than have been observed from merely thinking aloud without an audience.

(From Wikipedia)