CSCI 132:
Basic Data Structures and Algorithms

References, Static Methods, Exceptions

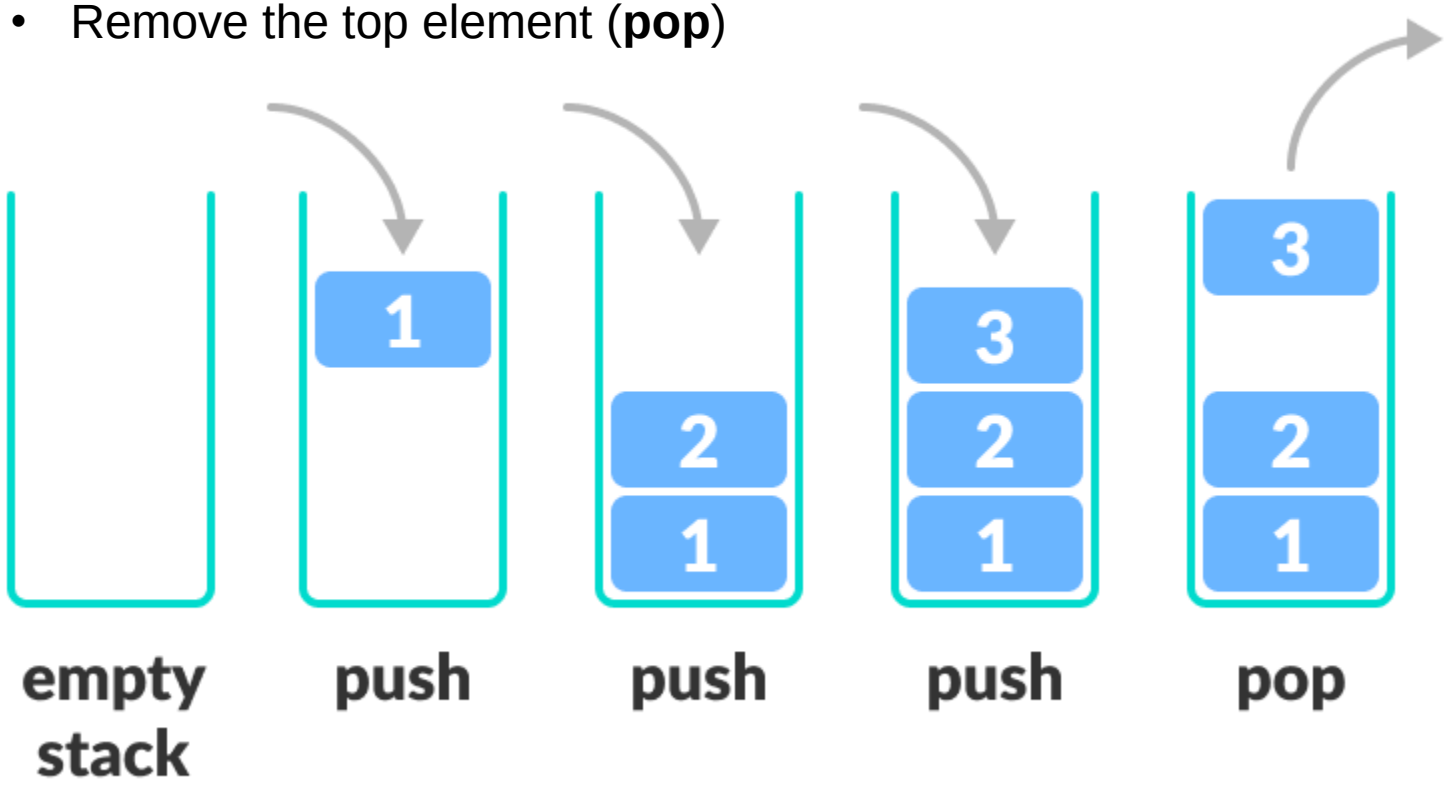
Reese Pearsall

Spring 2025

A **stack** is a data structure that can hold data, and follows the **last in first out (LIFO)** principle

We can:

- Add an element to the top of the stack (**push**)
- Remove the top element (**pop**)



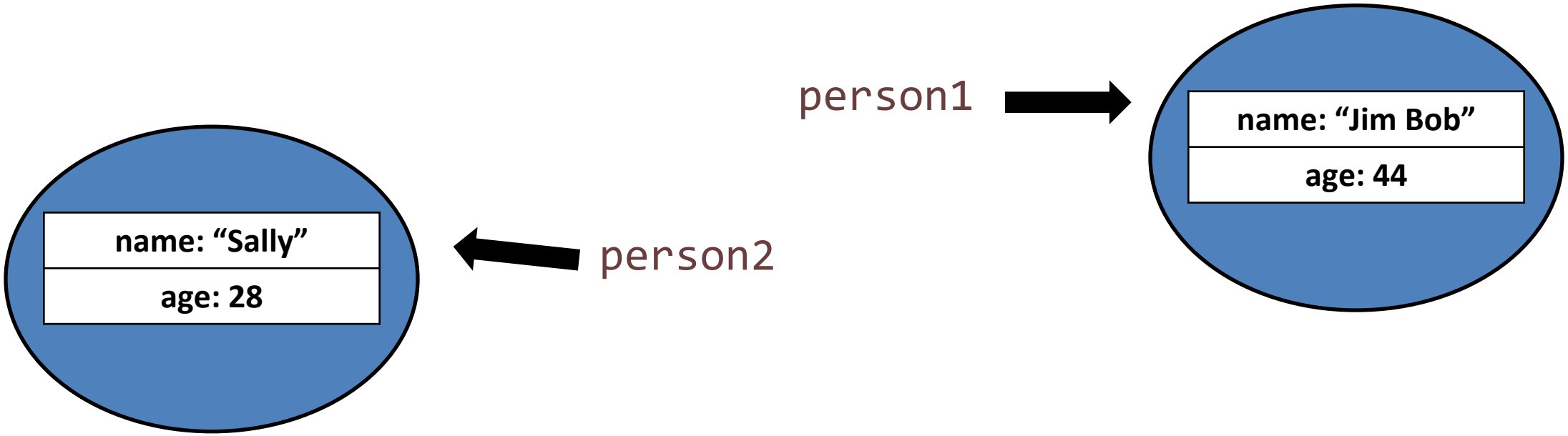
Any Stack (Stack ADT) must be able to:

- `Push()`
- `Pop()`
- `Peek()`
- `isEmpty()`

We can implement a Stack using an **Array**, or a **linked list**

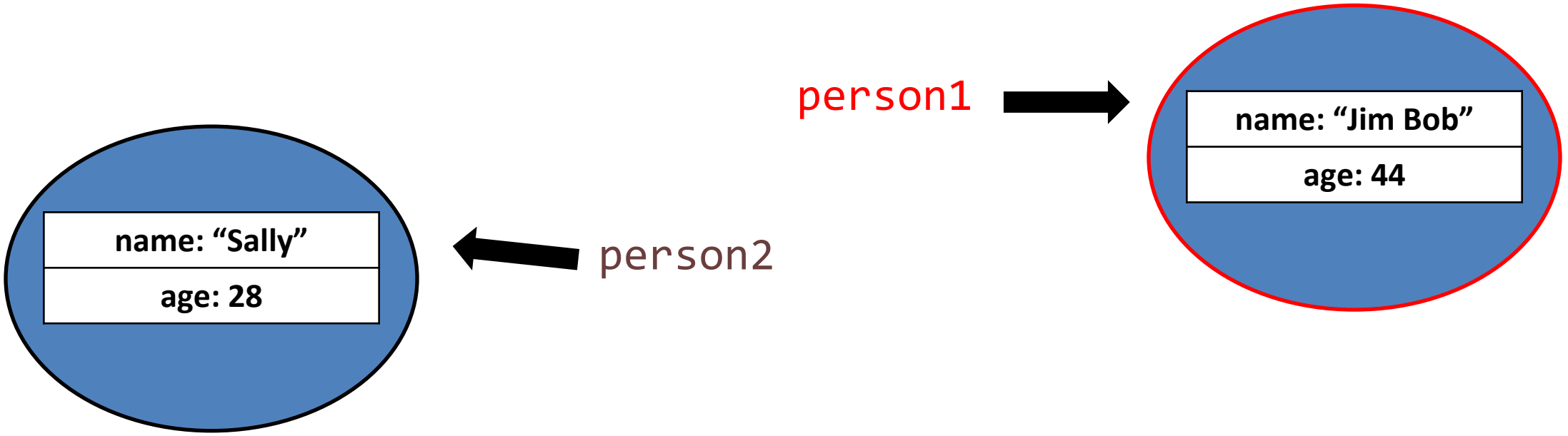
```
public class ReferencesDemo {
        public static void main(String[] args) {

                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");

        }
}
```

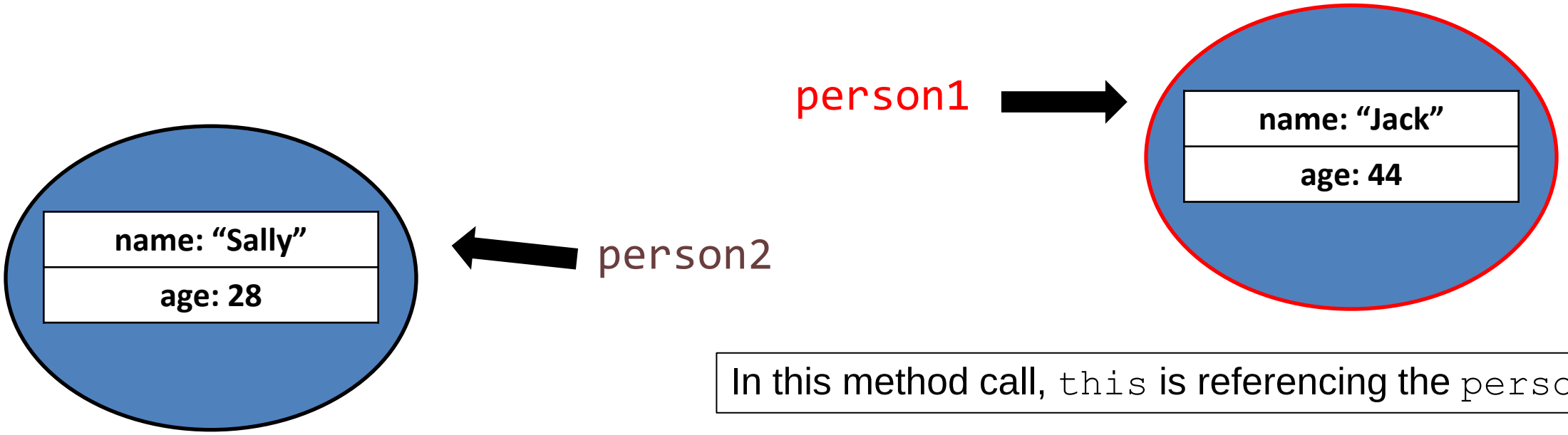
`person1` and `person2` are **references** to a Person object



```
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 **references** to a Person object

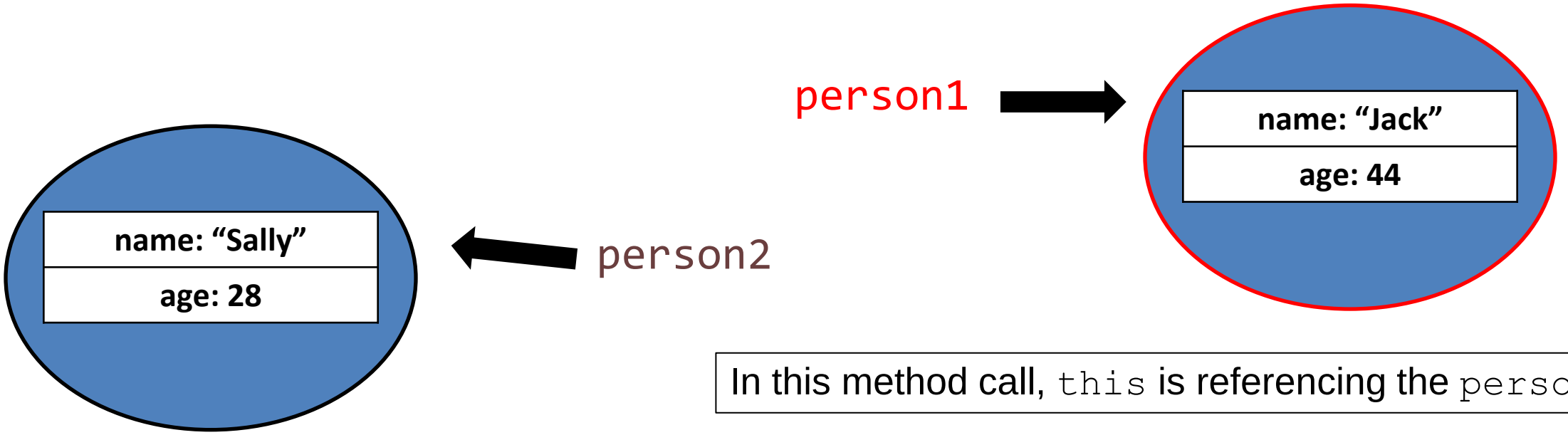


In this method call, `this` is referencing the `person1` object

```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;
	}
}
```

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

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



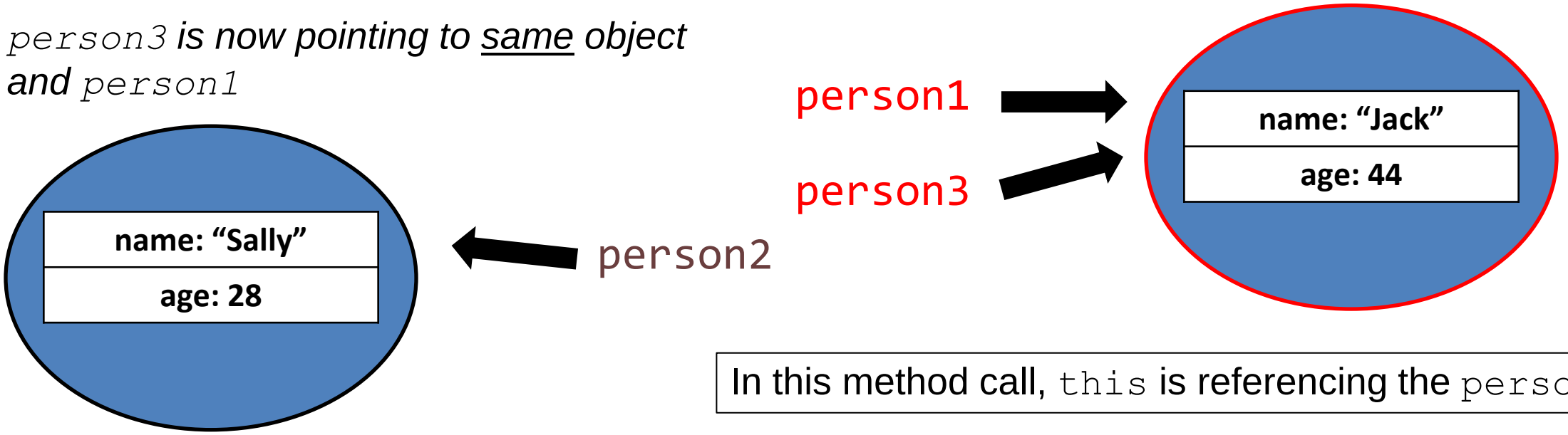
In this method call, `this` is referencing the `person1` object

```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;
	}
}
```

```java
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`



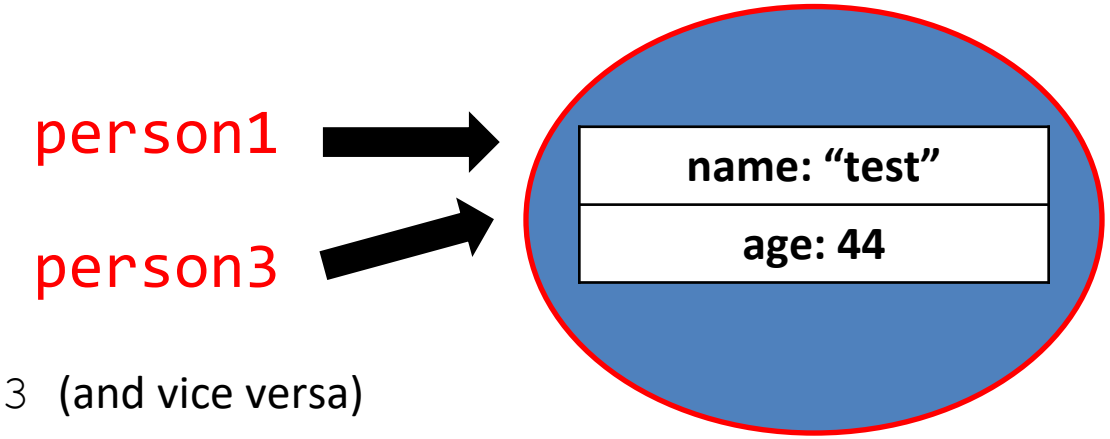
In this method call, `this` is referencing the `person1` object

```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;
                person1.changeName("test");
        }
}
```

```java
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`



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

```
System.out.println(person1.getName())
System.out.println(person3.getName())
```

→ “test”
→ “test”

Static methods are methods in Java that can be called without creating an object of a class

```
public class StaticDemo {
        public static void fun1(String arg1) {
                System.out.println(arg1);
        }
        public static void main(String[] args) {
                fun1("Hello");
        }
}
```

I do **not** need to create a `StaticDemo` object in order to call the `fun1()` method

Static methods are methods in Java that can be called without creating an object of a class

```java
public class StaticDemo {
        public static void main(String[] args) {
                AnotherClass.funMethod("Hello");
        }
}
```

StaticDemo.java

```java
public class AnotherClass {
    public static void funMethod(String arg)
    {
        System.out.println(arg);
    }
}
```

AnotherClass.java

If the static method is in another class, we can access it by giving the class name (`AnotherClass`)

Once again, I do not need to create an `AnotherClass` object to call this static method

However, now objects are no longer an implicit argument to this method (cant use `this` anymore)

Static methods are methods in Java that can be called without creating an object of a class

```
Error: static method cannot be referenced from a non static context
```

```
funMethod("Hello");
```

This is a very common error to see in Java.

- You can turn the method static by adding the static keyword in the method definition *(Easy and quick fix)*
- Or you use OOP and call the method on an instance of the class

```
AnotherClass obj = new AnotherClass()
obj.funMethod("Hello")
```

(Usually this is the better solution 80% of the time)

`try/catch` and **`exceptions`** are a way to run a piece of code (“try”), and then deal (“catch”) with errors

It will execute the body of **`try`**, and if a certain error/exceptions arises, then it will run the body of the **`catch`** statement

You can catch any error, or a specific error (FileNotFound, ArrayIndexOutOfBounds, NullPointerException)

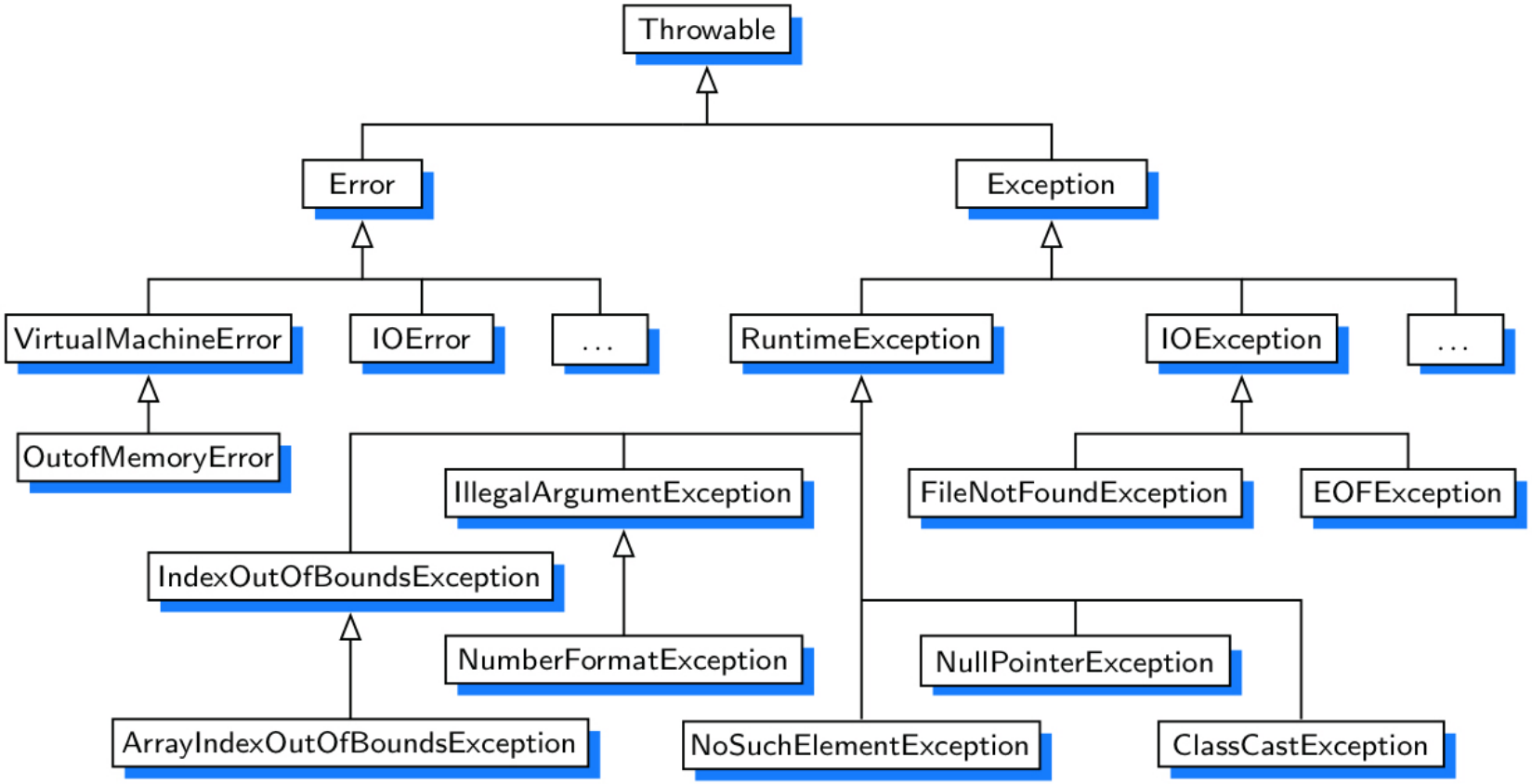


Figure 2.7: A small portion of Java's hierarchy of Throwable types.

```
while(userChoice != 3) {
        try {
                Scanner scanner = new Scanner(System.in);
                userChoice = scanner.nextInt();
                if(userChoice == 1) {
                        System.out.println("Hello");
                }
                else if(userChoice == 2) {
                        System.out.println("Hey");
                }
                else if(userChoice == 3) {
                        System.out.println("Goodbye");
                }
                else {
                        System.out.println("Enter a valid integer");
                }
        }
        catch(Exception e) {
                System.out.println(e);
                System.out.println("Invalid input detected. Please try again");
        }
        printOptions();
```

Java will **try** to execute this block of code

```
while(userChoice != 3) {
        try {
                Scanner scanner = new Scanner(System.in);
                userChoice = scanner.nextInt();
                if(userChoice == 1) {
                        System.out.println("Hello");
                }
                else if(userChoice == 2) {
                        System.out.println("Hey");
                }
                else if(userChoice == 3) {
                        System.out.println("Goodbye");
                }
                else {
                        System.out.println("Enter a valid integer");
                }
        }
        catch(Exception e) {
                System.out.println(e);
                System.out.println("Invalid input detected. Please try again");
        }
        printOptions();
```

Java will **try** to execute this block of code

If any error happens, instead of crashing, it will execute the body of the **catch**