Assess Yourself

Practice Question #1
What does it mean for an object to be *immutable*?

Practice Question #2
How can we *make* an object immutable?

Practice Question #3
Why do we generally make instance variables private?

Practice Question #4
Give an example that shows *when/why* we want to make instance variables private.

Assess Yourself

Practice Question #1

What does it mean for an object to be *immutable*?

An object is immutable if none of its instance variables can be changed/mutated after it is created.

Assess Yourself

Practice Question #2
How can we *make* an object immutable?

Objects can be made immutable by making all attributes private, having no setters for any attributes, and only returning copies of (immutable) attributes.

Assess Yourself

Practice Question #3

Why do we generally make instance variables private?

We make instance variables private as a means of access control; preventing outside objects from manipulating instance variables in undesired ways.

Assess Yourself

Practice Question #4
Give an example that shows *when/why* we want to make instance variables private.

In the case of a movie database, we may have the following situation:
```
movie.name = "F**king awful";
```

In order to prevent inappropriate names from being assigned to objects, we instead implement access control:
```
movie.setName("F**king awful");
```

With access control we can now do nothing, or throw an exception, if invalid/inappropriate data is entered.

SWEN20003
Object Oriented Software Development

Review and Slick Intro

Semester 1, 2019

The Road So Far

- OOP and Java Foundations
- Classes and Objects
  - Privacy and Immutability

Lecture Objectives

After this lecture you will be able to:

- Consolidate the last 3 weeks
- Develop reasonably complex OO systems
- Correctly use access control techniques
- Better explain privacy leaks
- Make a simple Slick game

Questions

Privacy

Can you have private methods?

Yes. A private method is used to "help" a class perform an action, and is generally not useful to any other classes.

Privacy

Can you have private classes?

Yes. You can have classes defined inside other classes (called "inner classes") and these can be private. Again, they usually exist to "help" the class, and are not useful to other classes.

Review

Example #1

```
public class Movie {

    private String title;

    public Movie(String title) {
        this.title = title;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
}
```

Example #1

```
public class Member {

    private Movie favouriteMovie;

    public Member(Movie favouriteMovie) {
        this.favouriteMovie = favouriteMovie;
    }

    public Movie getFavouriteMovie() {
        return favouriteMovie;
    }

    public void setFavouriteMovie(Movie favouriteMovie) {
        this.favouriteMovie = favouriteMovie;
    }
}
```

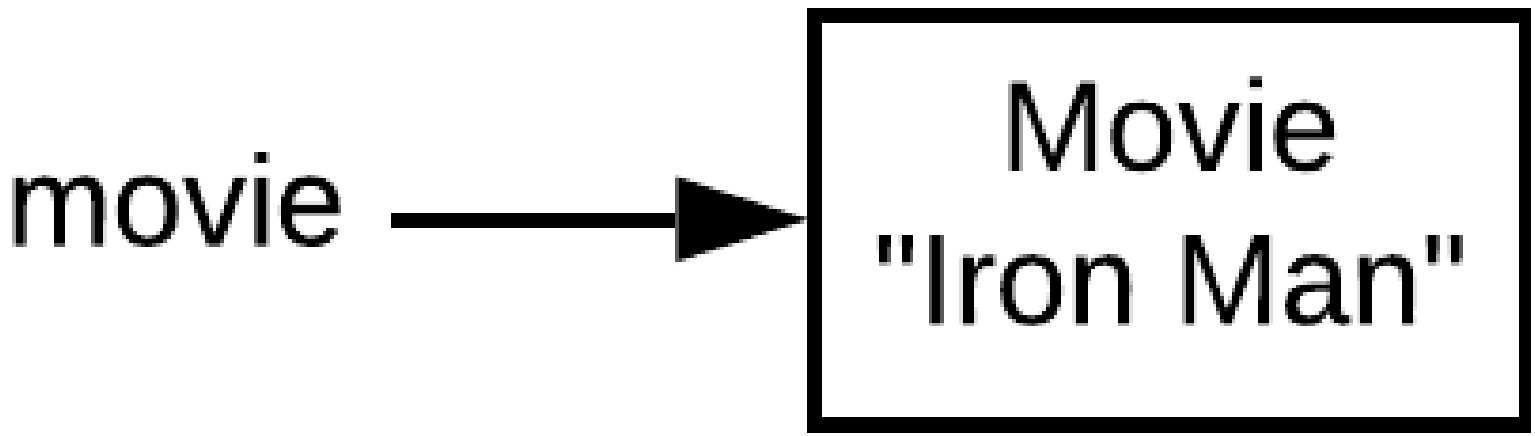
Example #1

```
Movie movie = new Movie("Iron Man");
Member matt = new Member(movie);
System.out.println(matt.getFavouriteMovie().getTitle());

Member ellie = matt;
ellie.setFavouriteMovie(new Movie("Copper Man");
System.out.println(matt.getFavouriteMovie().getTitle());
```

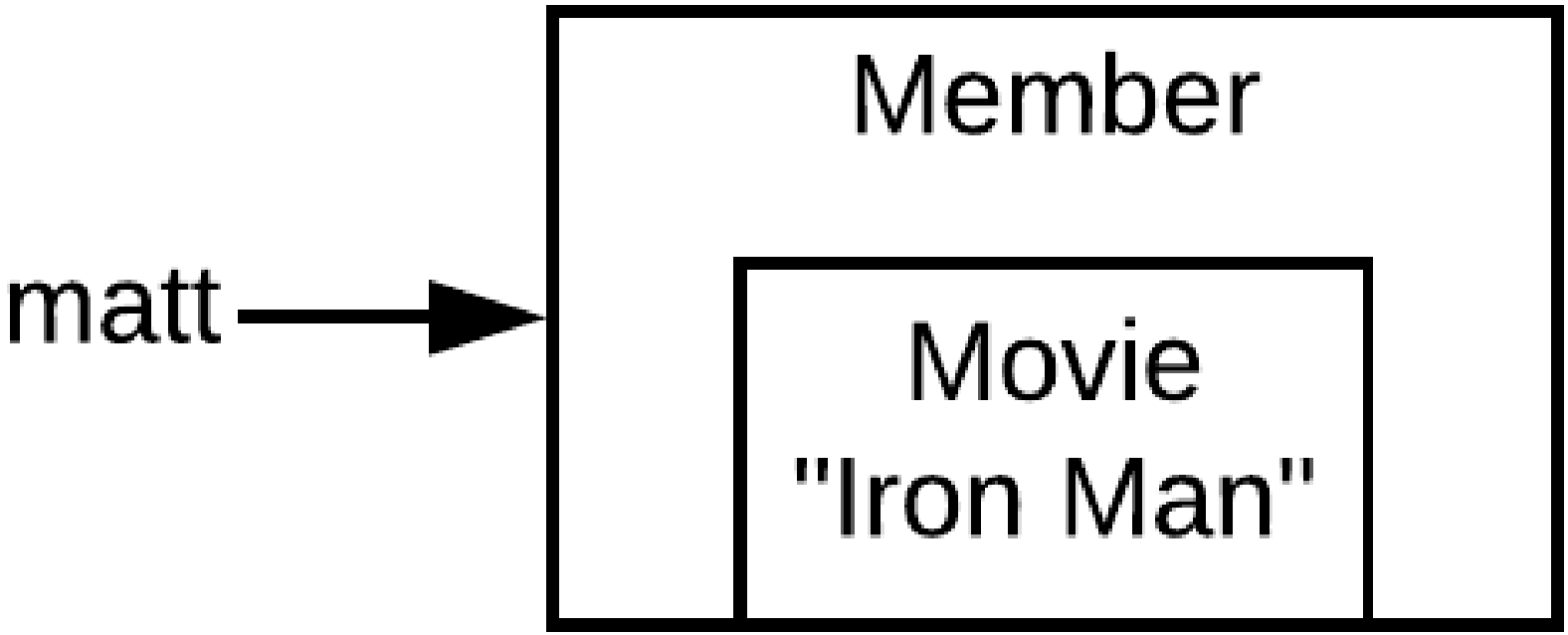
Example #1

```
Movie movie = new Movie("Iron Man");
```



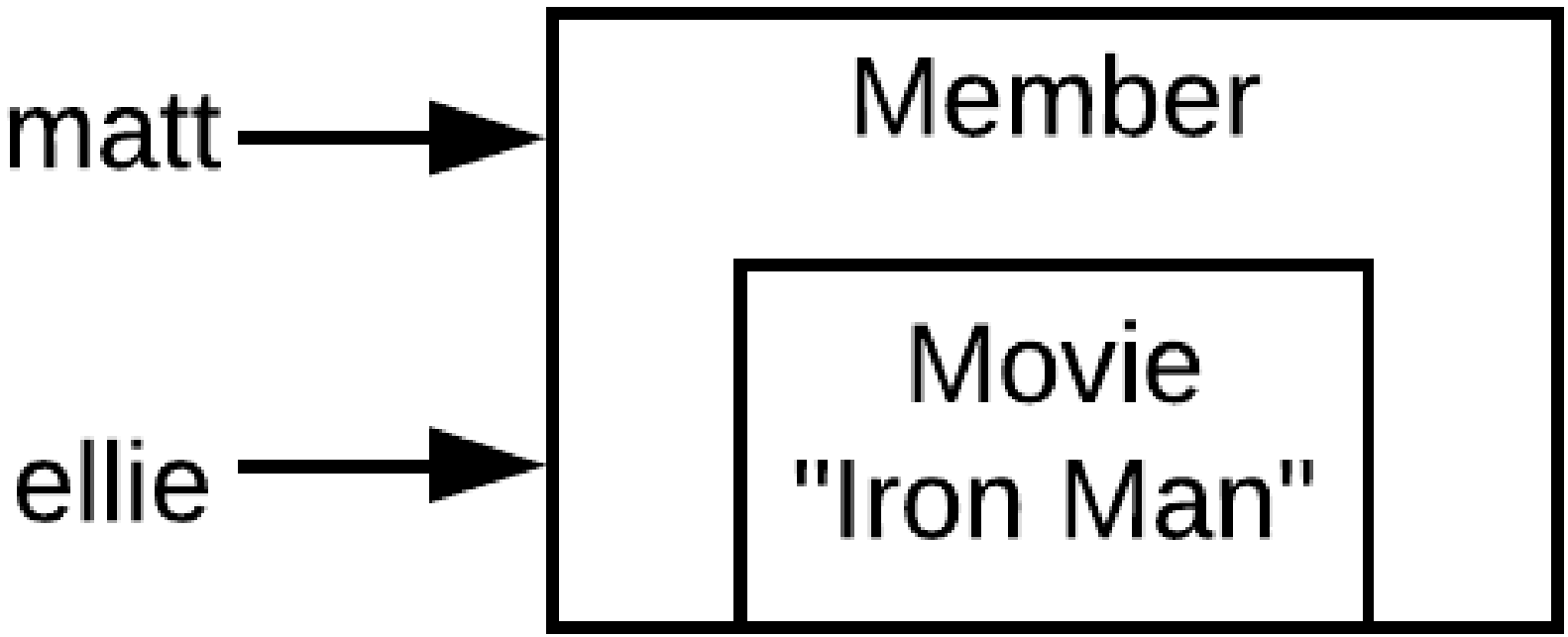
Example #1

```
Member matt = new Member(movie);
```



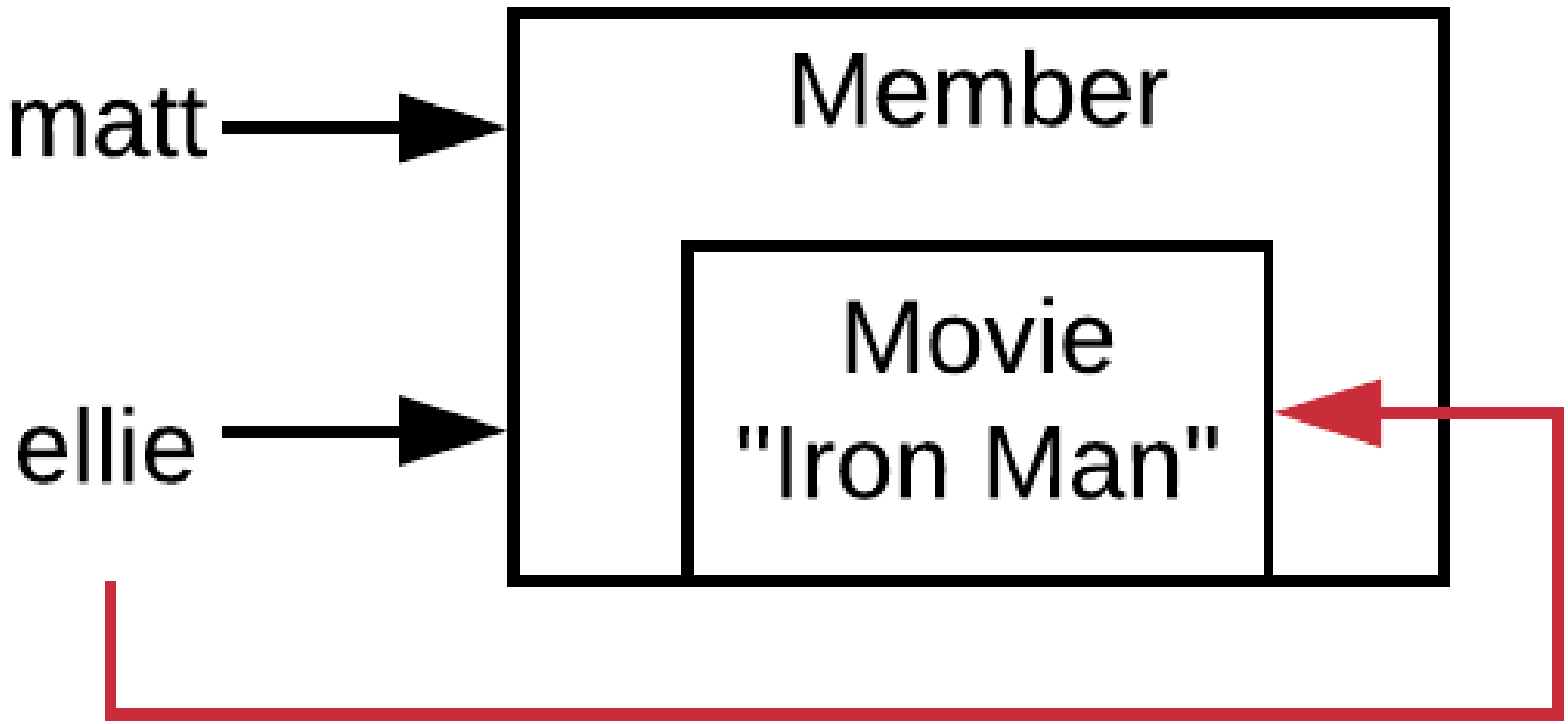
Example #1

```
Member ellie = matt;
```



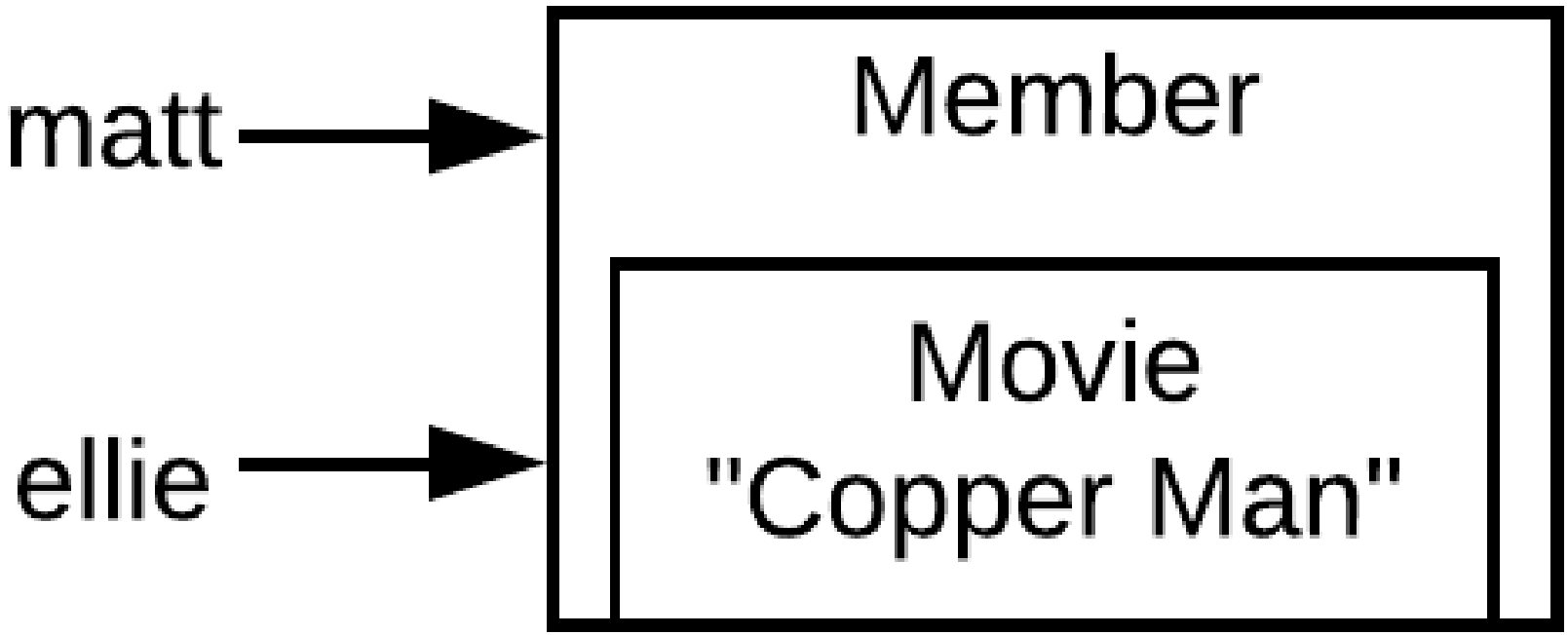
Example #1

```
ellie.setFavouriteMovie(...);
```



Example #1

```
ellie.setFavouriteMovie(new Movie("Copper Man");
```



Introducing the Copy Constructor

```
public class Member {

    private Movie favouriteMovie;

    public Member(Movie favouriteMovie) {
        this.favouriteMovie = favouriteMovie;
    }

    public Member(Member member) {
        this.favouriteMovie = member.favouriteMovie;
    }

    public Movie getFavouriteMovie() {
        return favouriteMovie;
    }

    public void setFavouriteMovie(Movie favouriteMovie) {
        this.favouriteMovie = favouriteMovie;
    }
}
```

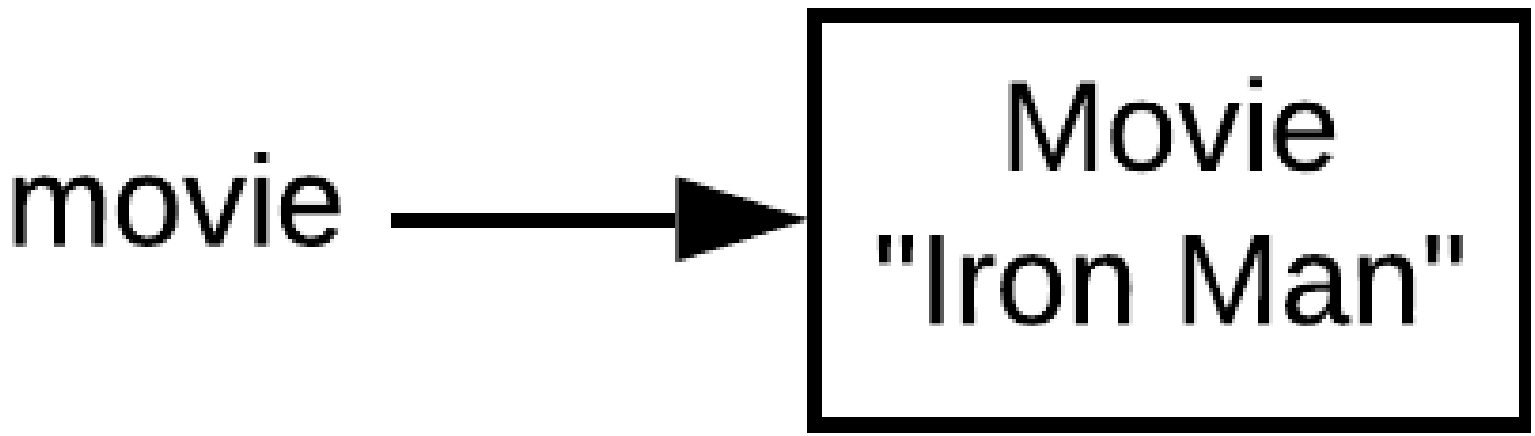
Example #2

```
Movie movie = new Movie("Iron Man");
Member matt = new Member(movie);
System.out.println(matt.getFavouriteMovie().getTitle());

Member ellie = new Member(matt);
Movie newMovie = ellie.getFavouriteMovie();
newMovie.setTitle("Copper Man");
System.out.println(matt.getFavouriteMovie().getTitle());
```

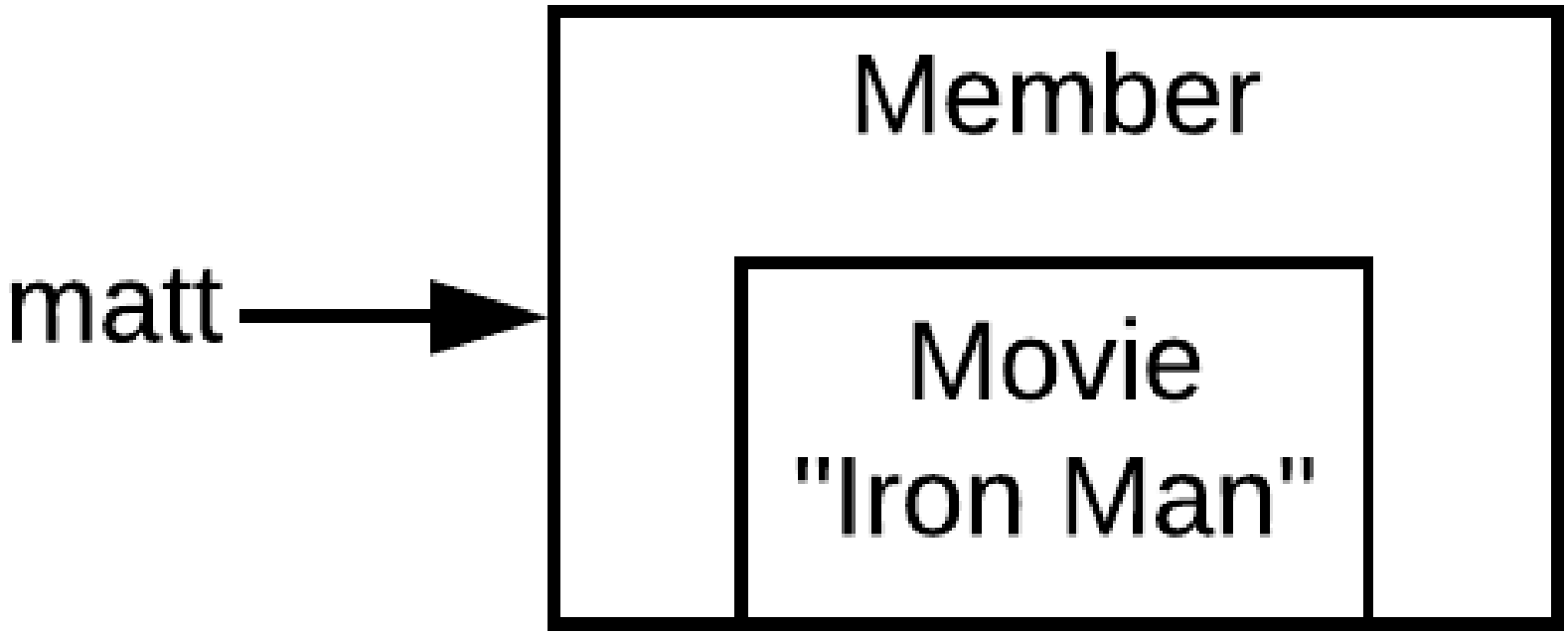
Example #2

```
Movie movie = new Movie("Iron Man");
```



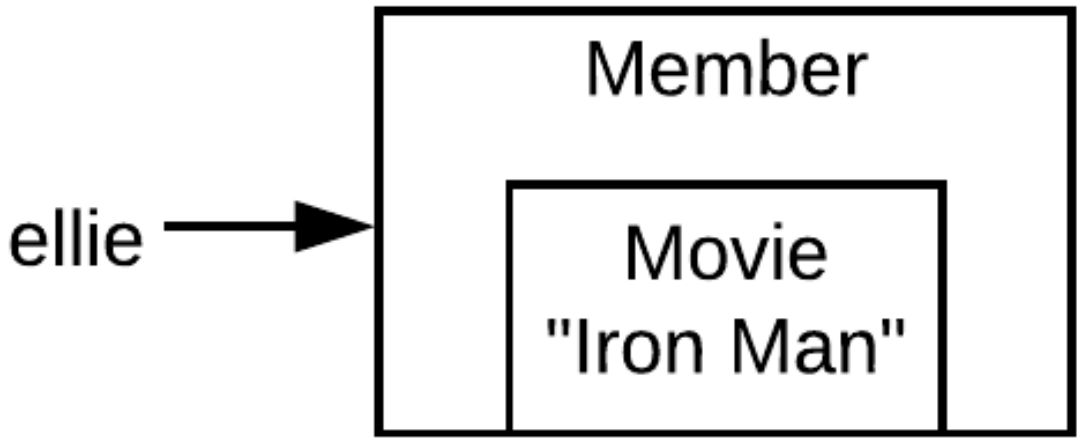
Example #2

```
Member matt = new Member(movie);
```



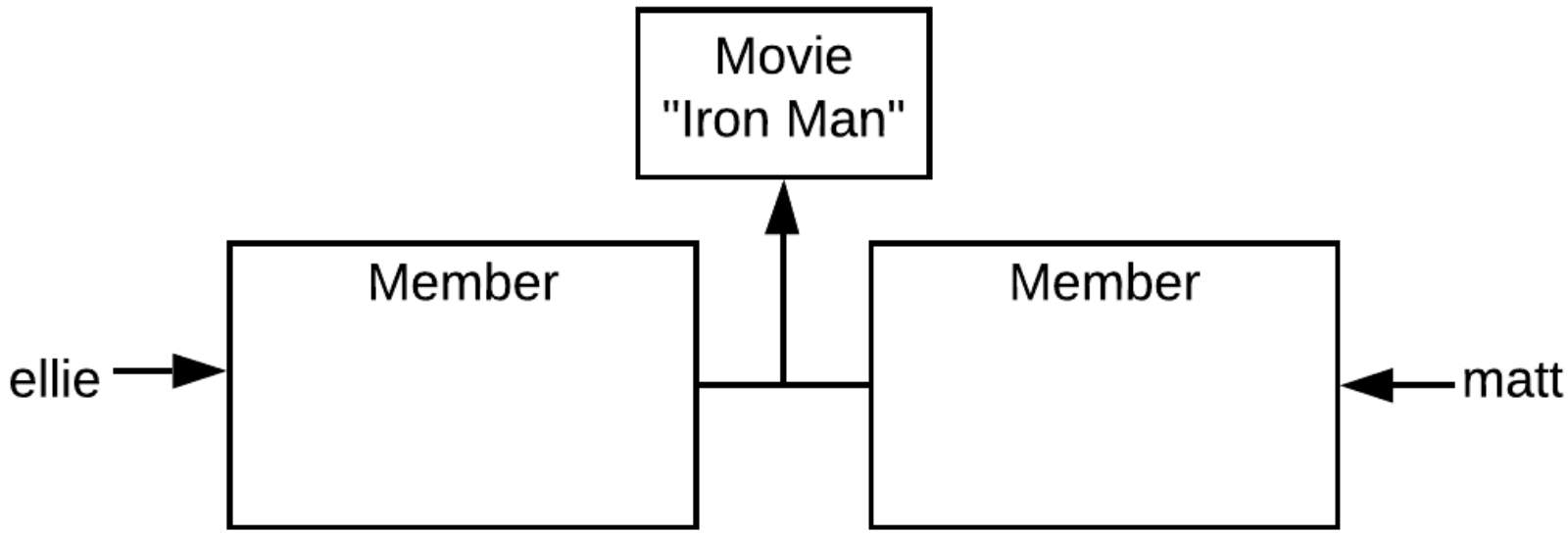
Example #2

```
Member ellie = new Member(matt);
```



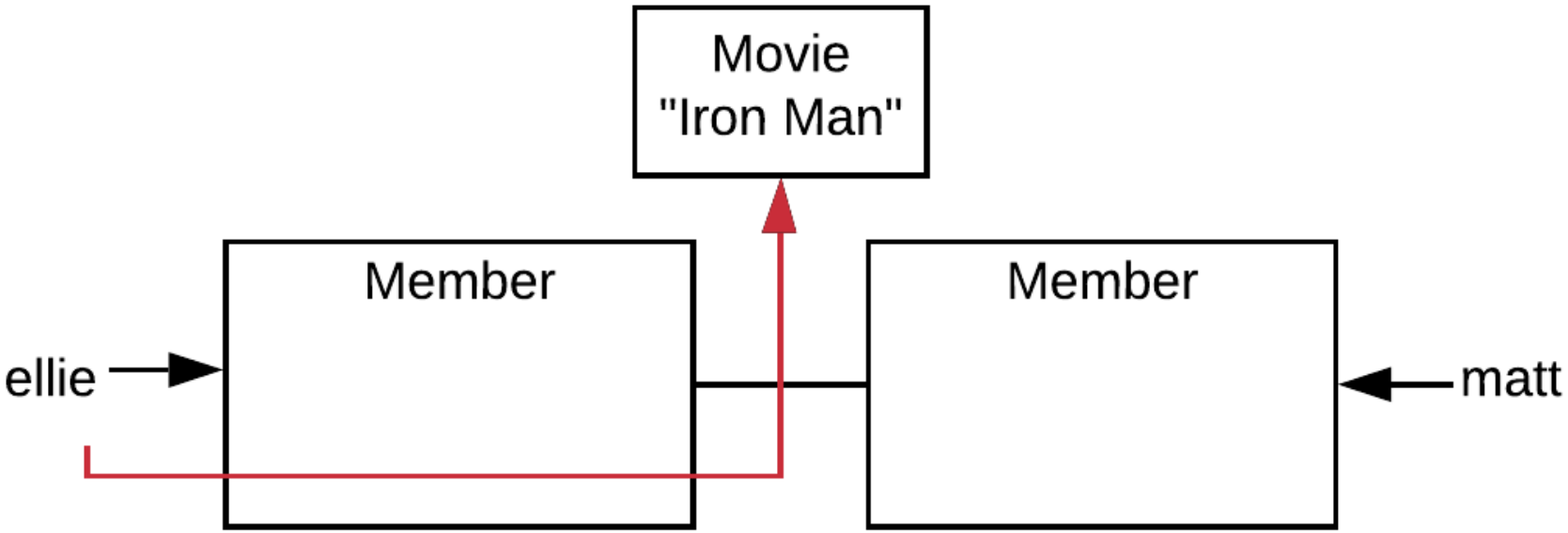
Example #2

```
public Member(Member member) {
    this.favouriteMovie = member.favouriteMovie;
}
```



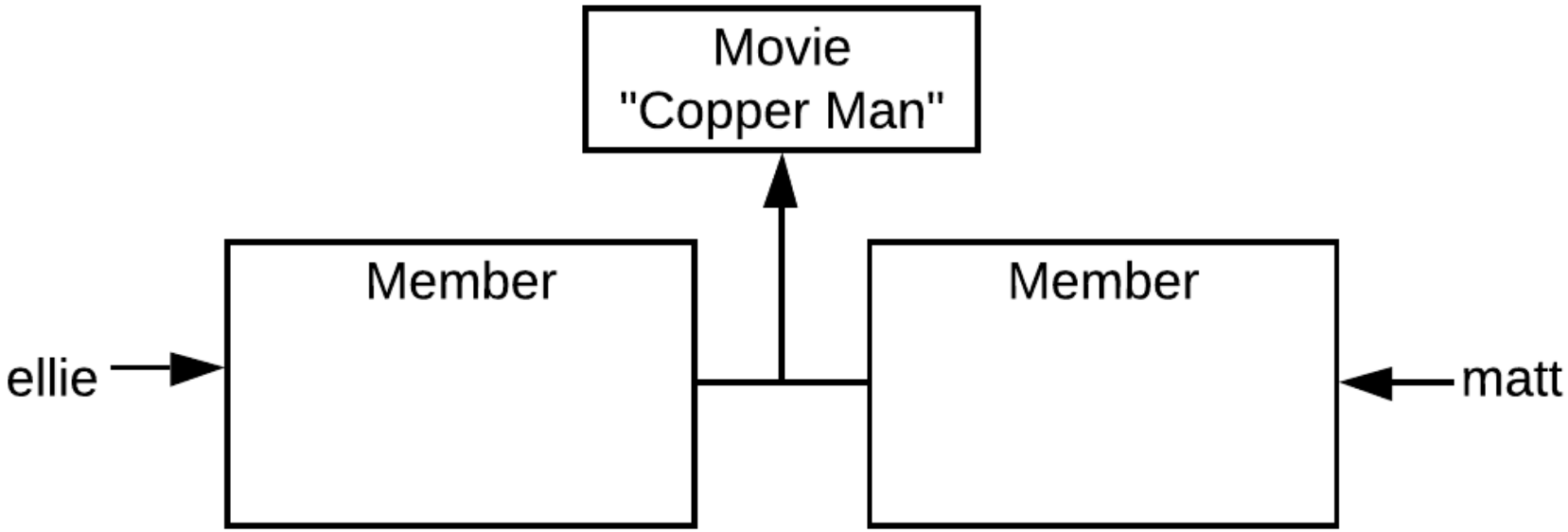
Example #2

```
Movie newMovie = ellie.getFavouriteMovie();
```



Example #2

```
newMovie.setTitle("Copper Man");
```



Example #3

When you play the game of thrones, you win or you die...

Every person in Game of Thrones has a name, an age, a house (or family), and a king/queen.

A house has a leader, some number of subjects, and a number of houses they are allied (friendly) with.

Design classes for this example.

Pitfall: Kings and Queens

What is wrong with our design?

- `Person`, `Leader`, `Subject`, `King`, and `Queen` are all different classes
- How different are they though?
- Are all people kings or queens?
- Are all kings or queens people?

Next lecture we will look at inheritance, which helps us solve this problem.

Example #3

Implement the `boolean isAlly(Person person)` for the `Person` class.

This method should return true if the person sent as an argument is an ally of the calling object.

For example:

```
System.out.println(jonSnow.isAlly(aryStark));
System.out.println(jonSnow.isAlly(cerseiLannister));
```

```
true
false
```

Example #3

```
public boolean isAlly(Person person) {
    for (House house : person.getHouse().getAllies()) {
        if (house.getName().equals(this.house.getName())) {
            return true;
        }
    }

    return false;
}
```

Example #4

The game of Pong involves two paddles on opposite sides of the screen, and a "ball" that bounces between them.

Each paddle is controlled by a separate player. The paddles can move up and down at 0.8px/ms.

The ball moves independently, and can move in any direction at 1.4px/ms. When the game starts, the ball starts in the centre of the screen and picks a random direction to start moving in.

If the ball collides with a paddle or exits the top and bottom sides of the screen, it "bounces off" and continues travelling.

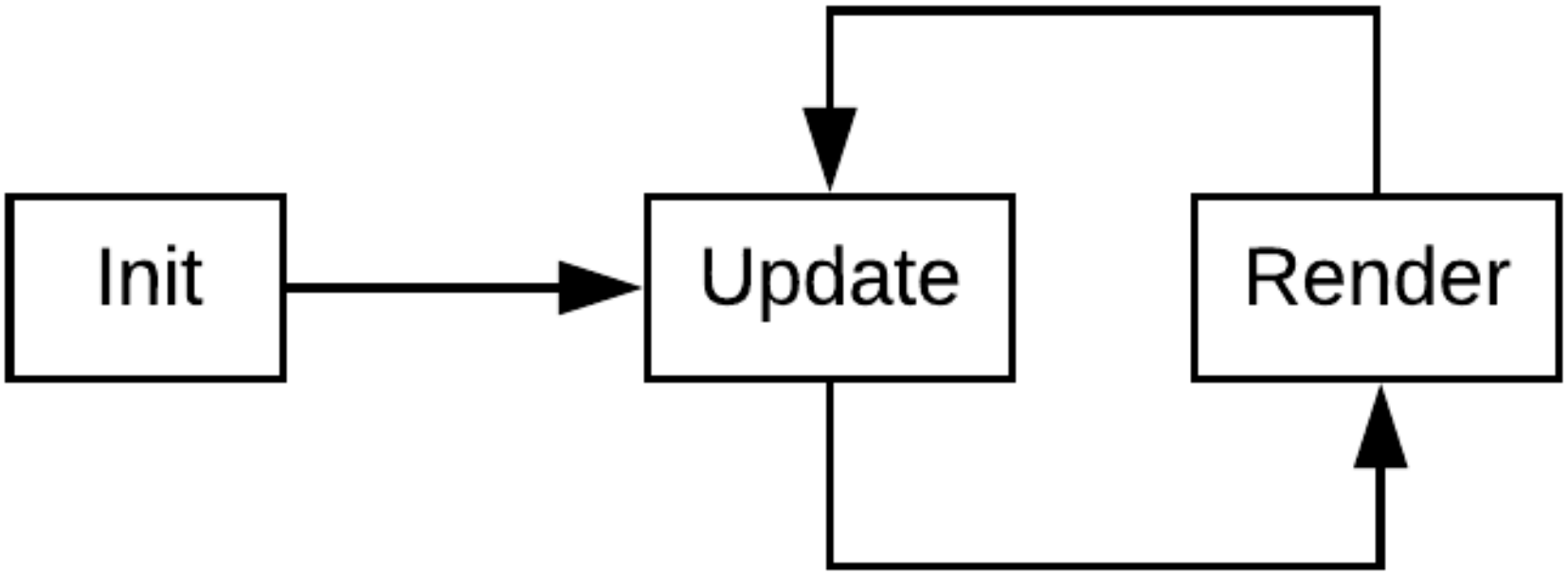
If the ball exists the left or right sides of the screen, the player on the opposite side scores a point, and the ball resets.

Slick

What is Slick?

- Java graphics library
- Built **on top of** the *Light Weight Java Gaming Library (LWJGL)*
- Capable of dealing with different kinds of input (keyboard, mouse, controllers) and output (images, graphics, sound)
- Has inbuilt functionality for cool things like particle effects, event listeners...
- **Disclaimer:** Not the best library... But the best we have

Slick Basics



Slick Basics

Keyword

init: Initialises the game and sets up any objects that are needed in order to play.

Keyword

update: Performs **computations** for objects in the game; acting on key presses, moving on the screen, etc.

Keyword

delta: The number of milliseconds since the last frame.

Keyword

render: Draws all objects to the screen. **That's it**.

Pitfall: Creating Objects

Don't. Create. Objects. In. Render.

Basic Slick Classes

Keyword

BasicGame: The mother of all games. We can either *use* this class, or *inherit* from it.

Keyword

Graphics: Allows us to draw objects (lines, rectangles, shapes) on the screen.

Keyword

Input: Allows us to detect and respond to key presses.

Full Slick documentation here.

Example #5

Let's implement our Pong design...

Metrics

Explain all the Slick keywords to someone. Your tutors will go through Slick with you again in the **week 5** workshop (starting this Friday).