Sample MMO

Lecture Question

Task: Write an Actor that creates other Actors

Ina package named actors, write classes ActorManager and ActorBot both of which extend Actor

- ActorManager should respond to message as follows
  - CreateActor: Creates a new ActorBot and adds it to the actor system. The id and after reference for the ActorBot constructor should be taken from the message. The created reference should be stored in a data structure
  - DestroyActor: Destroys the ActorBot with the given id and removes it from any data structures
  - Report: Sends the report message to all ActorBots
- ActorBot should have a constructor that takes a string (id) and an ActorRef and responds to:
  - Report: Send a Reporting message containing this Bots id (from the constructor) to the actor reference from the constructor

```scala
case class CreateActor(id: String, ref: ActorRef)
case class DestroyActor(id: String)
case object Report
case class Reporting(id: String)
```

Lab Overview

- In the GUIs lecture we made an app where a user could click on the GUI to place rectangles
- Use this GUI as a starting point and add database connections that will remember where all the rectangles were placed
- After restarting the app, all rectangles that would placed should still be on the GUI (loaded from the database)

MMO

- Massive Multiplayer Online (MMO) App
    - Typically refers to games, but can include any online app where many users interact in real time
- Today we'll see our first example where users are interacting in realtime in the same app
    - Excluding the chat app which technically met this condition

Example App

- We'll see a [simple] app where users can see each others mouse movements
  - Not particularly fun, but shows the technology used to build a full game

Example shown in IntelliJ

Creating Actors

- We saw that we can create a new actor system and add actors to it
- When we add an actor, we get a reference to that actor that can be used to send it messages

```
val system = ActorSystem("ActorSystem")

val one = system.actorOf(Props(classOf[Counter], "1"))

one ! Start
```

Creating Actors

- Sometimes we want actors to create more actors and add them to the system
- We can do this by accessing the actor's context
- Other syntax is the same as when creating an actor with the system directly

```
class ActorCreator() extends Actor {

  override def receive: Receive = {
    case CreateActor =>
      val ref = this.context.actorOf(Props(classOf[ActorType]))
  }

}
```

Destroying Actors

- To destroy an actor, send it the PoisonPill message
- Any actor with a reference to another actor can send it the PoisonPill message

```
actorRef ! PoisonPill
```

Lecture Question

Task: Write an Actor that creates other Actors

Ina package named actors, write classes ActorManager and ActorBot both of which extend Actor

- ActorManager should respond to message as follows
    - CreateActor: Creates a new ActorBot and adds it to the actor system. The id and after reference for the ActorBot constructor should be taken from the message. The created reference should be stored in a data structure
    - DestroyActor: Destroys the ActorBot with the given id and removes it from any data structures
    - Report: Sends the report message to all ActorBots
- ActorBot should have a constructor that takes a string (id) and an ActorRef and responds to:
    - Report: Send a Reporting message containing this Bots id (from the constructor) to the actor reference from the constructor

```
case class CreateActor(id: String, ref: ActorRef)
case class DestroyActor(id: String)
case object Report
case class Reporting(id: String)
```